Quantum Computing 102: Quantum Gates: Mathematical Representations and Properties

Kshipra Wadikar

March 12, 2025

1 Prerequisites

Quantum states are nothing but the vectors in a complex Hilbert space. In quantum computing, calculations are performed using quantum states. These states can be represented as vectors in a complex Hilbert space. To understand how to control these states with 'quantum gates,' we first need to learn about complex numbers, which are a key part of how these states work.

1.1 Complex Numbers and Representation

A complex number is expressed in the form: $z = a + ib,\ a, b \in \mathbb{R},\ \ i^2 = -1$.
The magnitude (or modulus) of a complex number $z = a + ib$ is given by: $|z| = \sqrt{a^2 + b^2}$.
The phase (or argument) of $z = a + ib$, denoted as θ, is the angle it makes with the positive real axis given by $\theta = tan^{-1}\left(\frac{b}{a}\right)$.
A complex number $z = a + ib$ can also be represented in polar form: $z = re^{i\theta}$ where $r = |z|,\ \theta$ is the phase and $e^{i\theta} = cos\theta + i\sin\theta,\ \ i^2 = -1$.
The **complex conjugate** of $z = a + ib$ is given by $z^* = a - ib$.

1.2 Matrices, Transpose, Hermitian Conjugate

A matrix is essentially a rectangular array of numbers. The transpose of a square matrix (a matrix with the same number of rows and columns) is formed by swapping its rows and columns. If A is a square matrix, its transpose is denoted as A^T.
For matrices containing complex numbers, we have the **Hermitian conjugate**, also known as the adjoint or dagger operation. This involves two steps:

1. Taking the complex conjugate of each element (changing the sign of the imaginary part). If A is a matrix, this results in A^*.
2. Taking the transpose of the resulting matrix, so $A^\dagger = (A^*)^T$.

This operation is denoted as $A^\dagger$.
For example, if $A = \begin{bmatrix} 1+i & 3 \\ 2+i & 4-i \end{bmatrix}$ then $A^\dagger = \begin{bmatrix} 1-i & 2-i \\ 3 & 4+i \end{bmatrix}$.

1.3 Unitary and Hermitian Matrices

A matrix U is **unitary** if $U^\dagger U = UU^\dagger = I$ where I is the identity matrix.
A matrix H is **Hermitian** if $H^\dagger = H$.

2 Quantum Gates

Analogous to classical logic gates, **quantum gates** are the fundamental building blocks of quantum circuits.
While classical gates manipulate the bits, quantum gates operate on **qubits** using **unitary matrices**.
A quantum gate acting on an $n-$qubit system is mathematically represented by a unitary matrix U of size $2^n \times 2^n$ and this matrix must satisfy the unitary condition $U^\dagger U = UU^\dagger = I$ where

1. U represents the quantum gate.
2. $U^\dagger$ (also written as U^H) is the Hermitian conjugate (complex conjugate transpose) of U.
3. I is an identity matrix of size $2^n \times 2^n$.

The unitary property ensures that quantum gates preserve probability and are reversible, making them fundamentally different from classical logic gates. Through unitary operations, quantum gates manipulate qubits and allow us to perform operations such as **superposition, entanglement**, and **measurement**. Below we analyze some fundamental quantum gates, their matrix representations and their effects on qubits.

2.1 Categorization of Quantum Gates

Quantum gates can be broadly classified as below.

1. Single-Qubit Gates (Pauli gates, Hadamard, Phase gates)
   Here our basis vectors are $|0\rangle = \begin{bmatrix}1\\0\end{bmatrix}$, $|1\rangle = \begin{bmatrix}0\\1\end{bmatrix}$.
2. Two-Qubit Gates (CNOT, SWAP, Controlled gates)
3. Multi-Qubit Gates (Toffoli, Fredkin)

We begin with the single qubit gates.

3 Single-Qubit Gates (Pauli gates, Hadamard, Phase gates)

1. **Pauli gates**
   Pauli-X (Bit Flip) Gate is defined as $X = \begin{bmatrix}0 & 1\\1 & 0\end{bmatrix}$.
   Here $X^\dagger = X$ and $X^\dagger X = XX^\dagger = I_{2\times 2}$. (unitary property)
   $X|0\rangle = \begin{bmatrix}0 & 1\\1 & 0\end{bmatrix}\begin{bmatrix}1\\0\end{bmatrix} = \begin{bmatrix}0\\1\end{bmatrix} = |1\rangle$ and
   $X|1\rangle = \begin{bmatrix}0 & 1\\1 & 0\end{bmatrix}\begin{bmatrix}0\\1\end{bmatrix} = \begin{bmatrix}1\\0\end{bmatrix} = |0\rangle$.
   The Pauli-X gate acts as a quantum analog of the classical NOT gate, flipping the state of a qubit.
   Pauli-Y (Bit and Phase Flip) Gate is defined as $Y = \begin{bmatrix}0 & -i\\i & 0\end{bmatrix}$.

Here $Y^\dagger = Y$ and $Y^\dagger Y = YY^\dagger = I_{2\times 2}$. (unitary property)

$Y|0\rangle = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ i \end{bmatrix} = i|1\rangle$ and

$Y|1\rangle = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} -i \\ 0 \end{bmatrix} = -i|0\rangle$.

Thus, the Pauli-Y gate applies a bit flip along with a phase shift of $\pm i$.

Pauli-Z (Phase Flip) Gate is defined as $Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$.

Here $Z^\dagger = Z$ and $Z^\dagger Z = ZZ^\dagger = I_{2\times 2}$. (unitary property)

$Z|0\rangle = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} = |0\rangle$ and

$Z|1\rangle = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ -1 \end{bmatrix} = -|1\rangle$.

The Pauli-Z gate introduces a phase shift of -1 to $|1\rangle$ but does not alter $|0\rangle$.

2. **Hadamard (Superposition) Gate** The Hadamard Gate (H gate) is very crucial for quantum computing because it creates superpositions.

   The Hadamard Gate (H gate) is defined as $H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$.

   Here $H^\dagger = H$ and $H^\dagger H = HH^\dagger = I_{2\times 2}$. (unitary property)

   $H|0\rangle = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$

   $H|1\rangle = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ -1 \end{bmatrix} = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$

   Thus, the Hadamard gate creates an equal superposition of $|0\rangle$ and $|1\rangle$.

3. **S gate** S gate is also known as the $\frac{\pi}{2}-$phase shift gate.

   The S gate is defined as $S = \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix}$.

   Here $S^\dagger = \begin{bmatrix} 1 & 0 \\ 0 & -i \end{bmatrix}$ and $S^\dagger S = SS^\dagger = I_{2\times 2}$. (unitary property)

   $S|0\rangle = \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} = |0\rangle$ and

   $S|1\rangle = \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ i \end{bmatrix} = i|1\rangle$.

4. **T gate** T gate is also known as as $\frac{\pi}{4}-$phase shift gate.

   The T gate is defined as $T = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{bmatrix}$.

   Here $T^\dagger = \begin{bmatrix} 1 & 0 \\ 0 & e^{-i\pi/4} \end{bmatrix}$ and $T^\dagger T = TT^\dagger = I_{2\times 2}$. (unitary property)

   $T|0\rangle = \begin{bmatrix} 1 & 0 \\ 0 & e^{\frac{i\pi}{4}} \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} = |0\rangle$ and

   $T|1\rangle = \begin{bmatrix} 1 & 0 \\ 0 & e^{\frac{i\pi}{4}} \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ e^{i\pi/4} \end{bmatrix} = e^{i\pi/4}|1\rangle$.

3.1 Summary of Single-Qubit Gates

Table 1: Summary of Single-Qubit Gates

Gate	Matrix Representation	Effect	Effect on State Vectors
Pauli-X	$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$	Bit-flip	$\|0\rangle \to \|1\rangle\,,\ \|1\rangle \to \|0\rangle$
Pauli-Y	$\begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}$	Bit-flip + Phase-flip	$\|0\rangle \to i\,\|1\rangle\,,\ \|1\rangle \to -i\,\|0\rangle$
Pauli-Z	$\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$	Phase-flip of $\|1\rangle$	$\|0\rangle \to \|0\rangle\,,\ \|1\rangle \to -\,\|1\rangle$
Hadamard	$\frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$	Creates superposition	$\|0\rangle \to \frac{1}{\sqrt{2}}(\|0\rangle + \|1\rangle), \quad \|1\rangle \to \frac{1}{\sqrt{2}}(\|0\rangle - \|1\rangle)$
S-gate	$\begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix}$	$\frac{\pi}{2}$ phase shift	$\|0\rangle \to \|0\rangle\,,\ \|1\rangle \to i\,\|1\rangle$
T-gate	$\begin{bmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{bmatrix}$	$\frac{\pi}{4}$ phase shift	$\|0\rangle \to \|0\rangle\,,\ \|1\rangle \to e^{i\pi/4}\,\|1\rangle$

4 Two-Qubit Gates (CNOT, SWAP, Controlled gates)

For a two-qubit system, the four basis states are formed using the Kronecker product.

$$|00\rangle = |0\rangle \otimes |0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \otimes \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$$|01\rangle = |0\rangle \otimes |1\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \otimes \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

$$|10\rangle = |1\rangle \otimes |0\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \otimes \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

$$|11\rangle = |1\rangle \otimes |1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \otimes \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

4.0.1 Single-Qubit Operations in a Two-Qubit System

Before introducing two-qubit gates, let us first examine how single-qubit gates act on individual qubits in a two-qubit system. **A single-qubit gate in a multi-qubit system only affects the qubit it is applied to, while the other qubits remain unchanged. To represent this mathematically, we use the tensor product of the gate with**

the identity matrix, which ensures that the other qubits remain unaffected. For a two-qubit system, applying a single-qubit gate G to:

- **Qubit 1 (Left Qubit, Most Significant Bit)**: $G_1 = G \otimes I_2$
- **Qubit 2 (Right Qubit, Least Significant Bit)**: $G_2 = I_2 \otimes G$

For example,

1. Apply Pauli-X gate to the first qubit of $|00\rangle$.
   It means apply Pauli-X gate to the first qubit of $|00\rangle$ and apply identity operation to the second qubit of $|00\rangle$.
   $X \otimes I_2 |00\rangle = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} = |10\rangle$ where I_2 is the 2×2 identity matrix.
   Interpretation: Applying Pauli-X to the first qubit flips $|00\rangle$ to $|10\rangle$ while the second qubit remains unchanged.

2. Apply Hadamard gate to the second qubit of $|01\rangle$.
   It means apply identity operation to the first qubit of $|00\rangle$ and apply Hadamard gate gate to the second qubit of $|01\rangle$.
   $I_2 \otimes H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & -1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ -1 \\ 0 \\ 0 \end{bmatrix} = \frac{1}{\sqrt{2}} \left(\begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ -1 \\ 0 \\ 0 \end{bmatrix} \right) = \frac{1}{\sqrt{2}} (|00\rangle - |01\rangle).$
   Interpretation: Applying Hadamard to the second qubit transforms $|01\rangle$ into an equal superposition of $|00\rangle$ and $|01\rangle$.

4.0.2 Single-Qubit Operations in a Higher-Qubit System

The same approach extends to higher-qubit systems. For an n-qubit system, applying a single-qubit gate G to the k-th qubit while leaving the others unchanged is achieved using the tensor product:

$$G_k = I_2^{\otimes(k-1)} \otimes G \otimes I_2^{\otimes(n-k)}$$

where:

1. $I_2^{\otimes(k-1)}$ represents the identity operation on the first $k-1$ qubits,
2. G is applied to the k-th qubit, and
3. $I_2^{\otimes(n-k)}$ ensures that the remaining qubits remain unchanged.

Example: Applying Hadamard to the third qubit in a three-qubit system

If we have a three-qubit system in the state $|000\rangle$ and want to apply a Hadamard gate only to the third qubit, we use:

$$I_2 \otimes I_2 \otimes H$$

which expands to:

$$(I_2 \otimes I_2 \otimes H) |000\rangle = \frac{1}{\sqrt{2}}(|000\rangle + |001\rangle).$$

Thus, the Hadamard gate is applied only to the third qubit, creating a superposition between $|000\rangle$ and $|001\rangle$ while leaving the first two qubits unchanged.
Conclusion: This method generalizes to any n-qubit system, allowing single-qubit operations to be applied independently to specific qubits while preserving the rest of the system.

4.0.3 Two-Qubit Gates (CNOT, SWAP, Controlled gates)

Single-qubit gates act on individual qubits. However, to build powerful quantum computers, we need gates that can make qubits interact. Two-qubit gates do this, enabling operations like entanglement and conditional logic.

4.0.4 CNOT (Controlled-X) Gate

The **Controlled-NOT (CNOT) gate**is a fundamental quantum gate that flips the state of the target qubit **only if** the control qubit is in the $|1\rangle$ state. It is defined by the unitary matrix:

$$\text{CNOT} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Effect on Basis States:

$$CNOT\,|00\rangle = |00\rangle\,, CNOT\,|01\rangle = |01\rangle\,, CNOT\,|10\rangle = |11\rangle\,, CNOT\,|11\rangle = |10\rangle.$$

This shows that when the control qubit (first qubit) is $|0\rangle$, the target qubit remains unchanged, but when the control qubit is $|1\rangle$, the target qubit flips.

4.0.5 SWAP Gate

The **SWAP Gate** interchanges the states of two qubits. It is represented by the unitary matrix:

$$\text{SWAP} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Effect on Basis States:

$$\text{SWAP}\,|00\rangle = |00\rangle\,, \text{SWAP}\,|01\rangle = |10\rangle\,, \text{SWAP}\,|10\rangle = |01\rangle\,, \text{SWAP}\,|11\rangle = |11\rangle.$$

This confirms that the **SWAP Gate swaps the states of two qubits** while leaving entangled states unchanged.

4.0.6 Controlled Gates (General Form)

A more generalized class of two-qubit gates is the **controlled gate**, where a unitary operation U is applied to the target qubit **only if** the control qubit is in the $|1\rangle$ state.

The matrix representation of a **Controlled-U gate** is:

$$\text{CU} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & u_{00} & u_{01} \\ 0 & 0 & u_{10} & u_{11} \end{bmatrix}$$

where U is a 2×2 unitary matrix applied to the target qubit.

For example: - The **Controlled-Z (CZ) gate** applies a Pauli-Z operation to the target qubit if the control qubit is $|1\rangle$:

$$\text{CZ} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}$$

The **Controlled-Hadamard (CH) gate** applies the Hadamard transform to the target qubit if the control qubit is $|1\rangle$.

4.0.7 Conclusion

Two-qubit gates like **CNOT, SWAP, and controlled gates** are essential for quantum computation, as they enable qubit interactions, entanglement, and conditional logic. These gates serve as building blocks for more complex quantum algorithms, including **quantum teleportation, superdense coding, and Grover's search algorithm**.

5 Multi-Qubit Gates (Toffoli, Fredkin)

In the previous section, we explored two-qubit gates, which allow interactions between qubits to perform entanglement and controlled operations. However, certain quantum algorithms require **multi-qubit gates** which operate on three or more qubits. Two of the most fundamental multi-qubit gates are the **Toffoli (CCNOT) gate** and the **Fredkin (CSWAP) gate**.

5.0.1 Toffoli Gate (Controlled-Controlled-NOT, CCNOT)

The **Toffoli gate**, also called the **Controlled-Controlled-NOT (CCNOT) gate**, is a three-qubit gate that flips the state of the third qubit (target qubit) **only if** the first two qubits (control qubits) are both in the $|1\rangle$ state. The matrix representation of the Toffoli gate is:

$$\text{CNOT} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

Effect on Basis States:

$$\text{CNOT}\,|000\rangle = |000\rangle,$$
$$\text{CNOT}\,|001\rangle = |001\rangle\,,$$
$$\text{CNOT}\,|010\rangle = |010\rangle\,,$$
$$\text{CNOT}\,|011\rangle = |011\rangle\,,$$
$$\text{CNOT}\,|100\rangle = |100\rangle\,,$$
$$\text{CNOT}\,|101\rangle = |101\rangle\,,$$
$$\text{CNOT}\,|110\rangle = |111\rangle\,, \quad \text{(Target qubit flips)}$$
$$\text{CNOT}\,|111\rangle = |110\rangle\,. \quad \text{(Target qubit flips)}$$

Interpretation:

1. If both control qubits are $|1\rangle$, the target qubit is flipped.
2. Otherwise, the state remains unchanged.
3. The Toffoli gate is important in **quantum error correction** and **reversible classical computing**.

5.0.2 Fredkin Gate (Controlled-SWAP, CSWAP)

The **Fredkin gate**, also known as the **Controlled-SWAP (CSWAP) gate**, is a three-qubit gate that **swaps the second and third qubits** if the first qubit (control qubit) is in the $|1\rangle$ state. It is represented by the following **unitary matrix**:

$$\text{CSWAP} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Effect on Basis States:

$$\begin{aligned} &\text{CSWAP}\,|000\rangle = |000\rangle\,, \\ &\text{CSWAP}\,|001\rangle = |001\rangle\,, \\ &\text{CSWAP}\,|010\rangle = |010\rangle\,, \\ &\text{CSWAP}\,|011\rangle = |011\rangle\,, \\ &\text{CSWAP}\,|100\rangle = |100\rangle\,, \\ &\text{CSWAP}\,|101\rangle = |110\rangle\,, \quad \text{(Swapped)} \\ &\text{CSWAP}\,|110\rangle = |101\rangle\,, \quad \text{(Swapped)} \\ &\text{CSWAP}\,|111\rangle = |111\rangle\,. \end{aligned}$$

Interpretation: - If the control qubit is $|0\rangle$, nothing changes. - If the control qubit is $|1\rangle$, the second and third qubits are swapped. - The Fredkin gate is useful in **reversible computing and quantum simulations**.

5.0.3 Conclusion

The **Toffoli and Fredkin gates** extend quantum logic beyond two-qubit operations and are crucial in **quantum algorithms, error correction, and reversible computing**.
The **Toffoli gate (CCNOT)** enables universal classical computation within quantum systems.
The **Fredkin gate (CSWAP)** helps implement controlled operations in quantum circuits.
These gates, along with the **CNOT, SWAP, and controlled gates**, form the essential toolkit for **constructing multi-qubit quantum circuits and algorithms**.

6 Measurement

Measurement is the process of extracting information from a quantum system. When we measure a qubit, its superposition collapses into one of the basis states, with certain probabilities.
For a single qubit $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$, the probability of measuring

- $|0\rangle$ is $|\alpha|^2$.
- $|1\rangle$ is $|\beta|^2$.

This extends to multi-qubit systems where measurement collapses the state into one of the computational basis states.

7 Quantum Circuit Implementation with Qiskit

7.1 Single-Qubit Gates with Qiskit

Let us implement basic single-qubit gates and observe their outputs.

1. transpile() is a crucial function in Qiskit that optimizes a quantum circuit for execution on a specific quantum hardware or simulator. It maps qubits, reduces depth, and ensures compatibility with a target backend.

2. **The Aer module**: It is a module in qiskit_aer that provides access to quantum simulators. It provides high-performance simulators for quantum circuits. Aer is used to simulate quantum circuits without real hardware. The actual class representing the simulator is AerSimulator.

3. **Statevector:** It represents the quantum state of a system as a vector.
   qiskit.quantum_info is a module in Qiskit that provides tools for analyzing and manipulating quantum states, channels, and operators.
   Statevector is a mathematical representation of quantum states, crucial for quantum information processing.

4. **Role of the Classical Bit**: To extract information from a qubit, we need to measure it and store the result. This measurement collapses the qubit state to either 0 or 1, and the result is stored in the classical bit.

The following code will create a quantum circuit, apply the Pauli-X gate, and then measure the result.

Listing 1: necessary imports

```
from qiskit import QuantumCircuit, transpile
from qiskit_aer import Aer  #  Use qiskit_aer for simulators
from qiskit.quantum_info import Statevector
```

In the following code, we apply the Pauli-X gate to $|0\rangle$ and measure it.

Listing 2: circuit with 1 qubit 1 classical bit

```
qc = QuantumCircuit(1, 1)#Create a quantum circuit with one
   qubit and one classical bit
state = Statevector.from_instruction(qc)
print("Initial State:", state)
```

The output of the above code is

```
Initial State: Statevector([1.+0.j, 0.+0.j],
        dims=(2,))
```

This is nothing but our $|0\rangle$.
Now, let us apply Pauli-X gate to qc.

Listing 3: Pauli-X gate

```
# Apply a Pauli-X gate
 qc.x(0)

# Get the statevector after applying Pauli-X gate
state = Statevector.from_instruction(qc)
print("State After Pauli-X gate:", state)
```

The output of the above code is

```
State After Pauli-X gate: Statevector([0.+0.j, 1.+0.j],
        dims=(2,))
```

Now, let us perform the measurement.

Listing 4: measurement

```
    # Measure the qubit
qc.measure(0, 0)

# Initialize Qiskit Aer simulator
simulator = Aer.get_backend('aer_simulator')

# Transpile circuit for execution
compiled_circuit = transpile(qc, simulator)

# Run the circuit using the Aer simulator
job = simulator.run(compiled_circuit)
result = job.result()

# Get measurement results
counts = result.get_counts()  #
print("Measurement Results:", counts)
```

The output of the above code is

```
Measurement Results: {'1': 1024}
```

Conclusion: The measurement results '1': 1024 confirm that the Pauli-X gate successfully flipped the initial qubit state $|0\rangle$ to $|1\rangle$. Since the simulation ran 1024 times, and all measurements yielded '1', we can conclude that the qubit was consistently in the $|1\rangle$ state after the gate was applied. This demonstrates the correct functionality of the Pauli-X gate in Qiskit.
The following code, will create a quantum circuit, apply the Hadamard gate, and then measure the result.

Listing 5: Hadamard gate and measurement

```
from qiskit import QuantumCircuit, transpile
from qiskit_aer import Aer  #
from qiskit.quantum_info import Statevector


# Create a quantum circuit with one qubit
qc = QuantumCircuit(1, 1)#Create a quantum circuit with one qubit
    and one classical bit

# Get the statevector before applying any gate
#Do NOT use Statevector(qc) directly on circuits
state = Statevector.from_instruction(qc)
print("Initial State:", state)

# Apply a Hadamard gate
qc.h(0)

# Get the statevector after applying Hadamard
state = Statevector.from_instruction(qc)
print("State After Hadamard:", state)

# Measure the qubit
qc.measure(0, 0)

# Initialize Qiskit Aer simulator
simulator = Aer.get_backend('aer_simulator')

# Transpile circuit for execution
compiled_circuit = transpile(qc, simulator)

# Run the circuit using the Aer simulator
job = simulator.run(compiled_circuit)
result = job.result()

# Get measurement results
counts = result.get_counts()  #
print("Measurement Results:", counts)
```

The output of the above code is

```
Initial State: Statevector([1.+0.j, 0.+0.j],
            dims=(2,))
State After Hadamard: Statevector([0.70710678+0.j, 0.70710678+0.j],
            dims=(2,))
Measurement Results: {'1': 506, '0': 518}
```

Conclusion: The output Measurement Results: '1': 506, '0': 518 shows that when measuring the qubit after applying the Hadamard gate, the results are roughly evenly split between '0' and '1'. This is consistent with the qubit being in an equal superposition, where there's an equal chance of measuring each outcome.

7.2 Fixing the Reverse Order Issue in Qiskit

Qiskit applies operations from left to right, but in matrix math, we describe qubits from left to right in ket notation. Thus, in a two-qubit system:

- Mathematically, a state is written as $|q_1 q_0\rangle$ (big-endian).
- In Qiskit, it is stored as $|q_0 q_1\rangle$ (little-endian).

Listing 6: Reverse qubit order to match textbook convention (big-endian)

```
from qiskit import QuantumCircuit
from qiskit.quantum_info import Statevector

# Create a 2-qubit quantum circuit
qc = QuantumCircuit(2)
qc.h(0)  # Apply Hadamard to qubit 0
qc.cx(0, 1)  # Apply CNOT with qubit 0 as control and qubit 1 as
   target

# Get statevector in Qiskit's order (little-endian)
state_qiskit = Statevector.from_instruction(qc)
print("Statevector (Qiskit order):")
print(state_qiskit)

# Reverse qubit order to match textbook convention (big-endian)
qc_reversed = qc.reverse_bits()
state_reversed = Statevector.from_instruction(qc_reversed)

print("\nStatevector (Reversed order, big-endian):")
print(state_reversed)
```

The output of the above code is

```
    Statevector (Qiskit order):
Statevector([0.70710678+0.j, 0.70710678+0.j, 0.        +0.j,
             0.        +0.j],
            dims=(2, 2))

Statevector (Reversed order, big-endian):
Statevector([0.70710678+0.j, 0.        +0.j, 0.70710678+0.j,
             0.        +0.j],
            dims=(2, 2))
```

reverse_qargs() also serves the same purpose.

- Use reverse_qargs() when working with statevectors to match the big-endian convention used in textbooks and theoretical papers.
- reverse_bits() is used when one needs to flip the entire circuit's qubit layout (useful when mapping to hardware).

7.3 Two-Qubit Gates with Qiskit

Listing 7: Two-Qubit Gates with Qiskit

```
    from qiskit import QuantumCircuit, transpile
from qiskit_aer import Aer
from qiskit.quantum_info import Statevector

# Create a quantum circuit with two qubits and two classical bits
qc = QuantumCircuit(2, 2)

# Get the initial statevector
state = Statevector.from_instruction(qc)
print("Initial State:", state)

# Apply a Hadamard gate to the first qubit
qc.h(0)
# Get the statevector after applying the gates
state = Statevector.from_instruction(qc)
print("State After H gate:", state)

# Reverse the qubit order for interpretation
reversed_state = state.reverse_qargs()
print("Statevector :", reversed_state)

print("\nStatevector (Big-endian order using reverse_qargs()):")
print(state_reversed)

# Apply a CNOT gate with the first qubit as control and the
   second as target
qc.cx(0, 1)

# Get the statevector after applying the gates
state = Statevector.from_instruction(qc)
print("State After Gates:", state)

# Reverse the qubit order for interpretation
reversed_state = state.reverse_qargs()
print("Statevector :", reversed_state)


# Measure the qubits
qc.measure([0, 1], [0, 1])  # Measure both qubits

# Initialize Qiskit Aer simulator
simulator = Aer.get_backend('aer_simulator')

# Transpile circuit for execution
compiled_circuit = transpile(qc, simulator)

# Run the circuit using the Aer simulator
job = simulator.run(compiled_circuit)
result = job.result()

# Get measurement results
```

```
counts = result.get_counts()
print("Measurement Results:", counts)
```

The output of the above code is

```
    Initial State: Statevector([1.+0.j, 0.+0.j, 0.+0.j, 0.+0.j],
            dims=(2, 2))
State After H gate: Statevector([0.70710678+0.j, 0.70710678+0.j, 0.        +0.j,
             0.        +0.j],
            dims=(2, 2))
Statevector : Statevector([0.70710678+0.j, 0.        +0.j, 0.70710678+0.j,
             0.        +0.j],
            dims=(2, 2))

Statevector (Big-endian order using reverse_qargs()):
Statevector([0.70710678+0.j, 0.        +0.j, 0.70710678+0.j,
             0.        +0.j],
            dims=(2, 2))
State After Gates: Statevector([0.70710678+0.j, 0.        +0.j, 0.        +0.j,
             0.70710678+0.j],
            dims=(2, 2))
Statevector : Statevector([0.70710678+0.j, 0.        +0.j, 0.        +0.j,
             0.70710678+0.j],
            dims=(2, 2))
Measurement Results: {'11': 529, '00': 495}
```

Conclusion: The measurement results confirm the entanglement, showing a near-equal probability of measuring the system in either the $|00\rangle$ or $|11\rangle$ state.

8 Quantum Circuit Implementation with PennyLane

Listing 8: Single-Qubit Gates with PennyLane

```
    import pennylane as qml
import numpy as np

# Initialize a PennyLane quantum device (statevector simulator)
dev = qml.device("default.qubit", wires=1, shots=None)  # Exact
   simulation

@qml.qnode(dev)
def quantum_circuit():
    return qml.state()  # Get initial statevector

@qml.qnode(dev)
def hadamard_circuit():
    qml.Hadamard(wires=0)
    return qml.state()  # Get statevector after Hadamard

@qml.qnode(qml.device("default.qubit", wires=1, shots=1000))  #
   Measurement simulation
def measurement_circuit():
    qml.Hadamard(wires=0)
```

```
    return qml.sample(qml.PauliZ(0))  # Measure in computational
        basis

# Get initial statevector
initial_state = quantum_circuit()
print("Initial State:", initial_state)

# Get state after Hadamard
final_state = hadamard_circuit()
print("State After Hadamard:", final_state)

# Get measurement results
measurements = measurement_circuit()
print("Measurement Results:", measurements[:10])  # Show first 10
    samples
```

The output of the above code is

```
Initial State: [1.+0.j 0.+0.j]
State After Hadamard: [0.70710678+0.j 0.70710678+0.j]
Measurement Results: [ 1.  1.  1.  1.  1.  1.  1. -1. -1. -1.]
```

Conclusion:

9 Conclusion

This document has provided a foundational overview of quantum gates, exploring their mathematical representations, properties, and implementations. We have examined single-qubit gates like the Pauli and Hadamard gates, two-qubit gates like the CNOT and SWAP gates, and multi-qubit gates such as the Toffoli and Fredkin gates. These gates, represented by unitary matrices, are the building blocks of quantum circuits, enabling quantum computation's unique capabilities. A crucial distinction between classical and quantum computation lies in the nature of their fundamental operations. Classical computation relies on a finite set of logic gates (e.g., AND, OR, NOT). In contrast, quantum gates are described by unitary matrices, and since there are infinitely many possible unitary matrices, there are, in effect, an infinite number of possible quantum gates. This vastness contributes significantly to the potential power and flexibility of quantum computation. The ability to perform a continuous range of transformations on qubits, rather than just discrete operations on bits, is a key factor in the potential quantum advantage. Furthermore, we explored the practical implementation of these gates using Qiskit, demonstrating how to construct and simulate quantum circuits. Understanding the mathematical representations and properties of quantum gates is essential for designing and analyzing quantum algorithms and ultimately harnessing the power of quantum computers.
In the next section of this series, we will explore quantum entanglement, a fundamental phenomenon that enables key quantum computing applications such as quantum teleportation and superdense coding. Understanding entanglement will further deepen our grasp of multi-qubit interactions and their role in quantum algorithms.