QUANTUM TELEPORTATION DEVELOPMENT AND DEPLOYMENT USING Q# LIBRARY AND C# CLIENT

Kayun Chantarasathaporn[1]*, Choonhapong Thaiupathump[2],
Nongnat Nopakun[3], Yong Li[4], Vasutan Tunbunheng[5] and Luo Yong[6]

Department of Information Technology and Management
Faculty of Business Administration[1,2,3]
International College[4,5] Language Institute[6]
Krirk University, 10220, Bangkok, Thailand[1,2,3,4,5]
Email: dr.kayun@gmail.com[1]*, choonhapong@gmail.com[2], nongnat.nop@staff.krirk.ac.th[3],
YL6787773688@yahoo.com[4], cyber_dragonet@yahoo.com[5], luo.yong@staff.krirk.ac.th[6]

Received: July 5, 2023
Revised: October 10, 2023
Accepted: October 24, 2023

Abstract

Quantum computing has been considered a new generation of computer. Its basic unit, the qubit, is different from that in a regular computer, which is referred to as a classical bit. While the classical bit's value can be either 0 or 1, a qubit can be both 0 and 1 at the same time. Quantum computing is not a general-purpose computer; it is appropriate for some specific solutions, especially complicated problems in many fields, such as biology, logistics, information security, etc. Two major principles that are essential for quantum computing study are entanglement and teleportation. The big obstacles faced by new researchers in quantum computing, for a long time, have been not understanding the related mathematics easily and not having a clear sample software development and deployment methodology that matches theorems explanations. This paper addresses these problems by explaining quantum teleportation in simplified mathematics, showing how to apply quantum logic gates in the solution, and creating a runnable quantum teleportation Q# software library, which is later called from a C# client program. Though the Q# sample code in this paper runs on a quantum simulator, it can be used in a real quantum computing system that is based on the same architecture, like Microsoft Azure Quantum, as well.

Keywords: Quantum, Teleportation, Entanglement, Simulator

1. Introduction

Quantum computing applies modern physics in both hardware and software architectures. In the past, most of the study cases were in theoretical research. After major world-class IT industry companies such as Microsoft, IBM, Google, and Amazon[1] joined the quantum computer world, the progression of both research and practical development has been accelerated. They have invested in both hardware and

software and shared the usage with the public at low or no cost. The platforms they provide are mostly based on cloud networks, which also connect to the real quantum computers in the backend. One of the vendors, Microsoft, also provides a quantum simulator that can run to prove quantum conceptual programs on regular personal computers.

Research in quantum computing is not just popular in the West; Eastern countries such as China, Japan, and India have also created many innovations[2]. In 2021, the University of Science and Technology of China (USTC) invented a light-based quantum computer that is much faster than using semiconductors[3].

One of the major differences between quantum computing and classical computing is the basic unit of processing[4]. Classical computers use bits, which are either 0 or 1, as their basic unit, while quantum computers use quantum bits or qubits, which can be 0, 1, or both 0 and 1 simultaneously. The state that can be both 0 and 1 simultaneously for a qubit is called superposition. With the capability of superposition, quantum computers can solve some very complicated tasks, which usually take much time for classical computers[5], in much shorter periods, such as encryption, optimization, logistic management, drug development, weather forecasting, etc.

Although there are many algorithms related to quantum computers, the fundamental concepts for this field are quantum entanglement and quantum teleportation. Both are related to mathematics. They can be explained in either complicated mathematics or simple matrices[6]. Basic mathematics and computer programming (in text and WebAPI platforms) related to quantum entanglement were mentioned in these articles[7][8]. This paper focuses on quantum teleportation, explaining it in plain mathematics and showing how to create a practical library for this concept that can be applied in several application platforms, including console, Windows, web applications, and WebAPI. Though there are many platforms and languages that can be used for developing applications[9][10], Q# and Microsoft quantum simulator are chosen since the solution can be created and run on regular personal computers, providing a good starting point for anyone..

The contents of this article are as follows: the difference between bits and qubits, basic quantum mathematics and logic gates, an overview of the quantum teleportation concept, the creation of a quantum teleportation library using the Q# language, a sample client program written in C# that works with the created library on the quantum simulator, and the results showing that the generated software solution can produce results as theorized. Although the solution provided in the current research runs on a quantum simulator, all codes can be migrated to run on a real quantum computer, in this case, Azure Quantum.

1.1 Fundamental of Quantum Computing

1.1.1 Classical Bit and Quantum Bit

Classical computers use bits, which are binary and can be either 0 or 1, as their basic unit. In contrast, quantum computers use quantum bits or qubits, which can represent both 0 and 1 simultaneously. For a single qubit, the state 0 is represented in Dirac notation as |0>, while |1> denotes the state 1. When a qubit is in a state of both 0 and 1 simultaneously, it is referred to as superposition. The Dirac notation for superposition of one qubit is $(\frac{|0>+|1>}{\sqrt{2}})$ as shown in Figure 1.

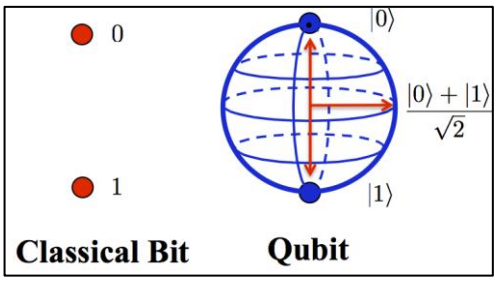


Figure 1 Classical Bit and Quantum Bit [11]

1.1.2 Dirac Notation and Basic Vector

A qubit is often written in Dirac notation; however, it can also be expressed in the form of a vector. Qubit 0, denoted as |0>, corresponds to $\binom{1}{0}$ in vector form (1 column matrix) while qubit 1, |1>, is represented as $\binom{0}{1}$. According to [7], it is shown that superposition of one qubit $(\frac{|0>+|1>}{\sqrt{2}})$, can be written in vector form as $\begin{pmatrix}\frac{1}{\sqrt{2}}\\ \frac{1}{\sqrt{2}}\end{pmatrix}$. For 2 qubits Dirac notation can be written as |0>|0> or |00>. By the way, the latter form is more popular. As expected, 2 qubits can also be written in vector form as demonstrated in [7].

Table 1 Dirac Notation and Matrix of 2 Qubits

2 Qubits	Dirac Notation	Vector (or Matrix)
qubits 0 and 0	|0>|0> or |00>	$\begin{pmatrix}1\\0\\0\\0\end{pmatrix}$
qubits 0 and 1	|0>|1> or |01>	$\begin{pmatrix}0\\1\\0\\0\end{pmatrix}$
qubits 1 and 0	|1>|0> or |10>	$\begin{pmatrix}0\\0\\1\\0\end{pmatrix}$
qubits 1 and 1	|1>|1> or |11>	$\begin{pmatrix}0\\0\\0\\1\end{pmatrix}$

1.2 Essential Quantum Logic Gates and Their Simplified Mathematical Verification

Instead of using mathematic equations, another prevalent approach in this field is the use of quantum logic gates or quantum gates. These gates consist of symbols for creating quantum circuits (quantum networks) as illustrated in Figure 4. Quantum gates are grounded in related mathematics and can be represented in the form of matrices or Dirac notations. Table 2 provides a summary of the essential gates and their equivalent matrices.

In this paper, which is related to quantum teleportation, the fundamental logic gates are CNOT, H, X, and Z gates. A brief explanation follows and will be further elaborated with related mathematics later.

1.2.1 Hadamard (H) gate and its own inverse property[6]

The H gate is used to change a qubit from a normal state (being either 0 or 1) to a superposition state (being both 0 and 1 simultaneously). It can also be considered as a 90° ($\pi/2$) rotation around the y-axis, followed by a 180° (π) rotation around the x-axis[12].

If a qubit is in a normal state, applying the H gate to it will result in a superposition state. On the other hand, if a qubit is already in superposition state, applying the H gate to it will return the qubit to the normal state, which can be considered similar to a classical bit. The following calculation will demonstrate the steps.

According to Table 2, the matrix for the H gate can be found in row 3 and column 4, represented as $\frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix}$, which is equivalent to $\begin{pmatrix}\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}}\\ \frac{1}{\sqrt{2}} & \frac{-1}{\sqrt{2}}\end{pmatrix}$. The results after applying H gate to qubit |0> and |1>, achieved by multiplying the H gate matrix with the qubit vectors, are shown in equations (1) and (2). Both states are in superposition.

$$H|0> = \begin{pmatrix}\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}}\\ \frac{1}{\sqrt{2}} & \frac{-1}{\sqrt{2}}\end{pmatrix}\begin{pmatrix}1\\0\end{pmatrix} = \begin{pmatrix}\frac{1}{\sqrt{2}}\\ \frac{1}{\sqrt{2}}\end{pmatrix} \quad (1)$$

$$H|1> = \begin{pmatrix}\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}}\\ \frac{1}{\sqrt{2}} & \frac{-1}{\sqrt{2}}\end{pmatrix}\begin{pmatrix}0\\1\end{pmatrix} = \begin{pmatrix}\frac{1}{\sqrt{2}}\\ \frac{-1}{\sqrt{2}}\end{pmatrix} \quad (2)$$

Next, the H gate's matrix will be multiplied with the qubits in the superposition state, and the result will be in the normal state, as shown in the output of equations (3) and (4) respectively.

$$H\begin{pmatrix}\frac{1}{\sqrt{2}}\\ \frac{1}{\sqrt{2}}\end{pmatrix} = \begin{pmatrix}\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}}\\ \frac{1}{\sqrt{2}} & \frac{-1}{\sqrt{2}}\end{pmatrix}\begin{pmatrix}\frac{1}{\sqrt{2}}\\ \frac{1}{\sqrt{2}}\end{pmatrix} = \begin{pmatrix}1\\0\end{pmatrix} = |0> \quad (3)$$

$$H\begin{pmatrix}\frac{1}{\sqrt{2}}\\ \frac{-1}{\sqrt{2}}\end{pmatrix} = \begin{pmatrix}\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}}\\ \frac{1}{\sqrt{2}} & \frac{-1}{\sqrt{2}}\end{pmatrix}\begin{pmatrix}\frac{1}{\sqrt{2}}\\ \frac{-1}{\sqrt{2}}\end{pmatrix} = \begin{pmatrix}0\\1\end{pmatrix} = |1> \quad (4)$$

With the above verification, the H gate can be considered as having its own inverse property. To work with the Hadamard gate easily, the state machine diagram shown in Figure 2 is often used.

1.2.2 Controlled-NOT (CNOT) gate toggles target bit if control bit is 1 [6]

The CNOT gate operates with two qubits. The first qubit is called the 'control qubit,' and the second qubit is called the 'target qubit.' If the control qubit is |1>, the value of the target qubit will be toggled (|0> => |1> or |1> => |0>). If the control qubit is |0>, the value of the target qubit will remain unchanged. This behavior can be observed in the truth table, specifically in row 4, column 3 of Table 2.

From row 4 column 4 of Table 2, the matrix form of the CNOT gate is as follows.

$$C = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

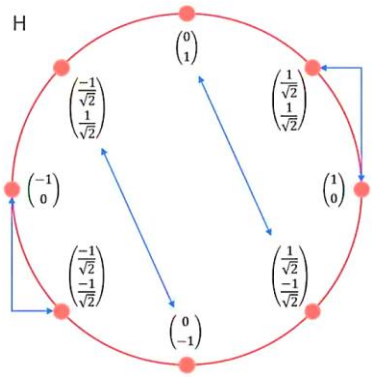


Figure 2 Hadamard (H) Gate's State Machine Diagram

(Source: https://tatourian.files.wordpress.com/2018/09/9.jpg)

To prove that the CNOT gate has the properties as mentioned above, the following demonstrates how it works. Additionally, two facts will be used in the proof: $\binom{1}{0}$ is the vector form of the qubit |0> and $\binom{0}{1}$ is the vector form of the qubit |1>.

- Case 1: Control qubit is 1. Target qubit will be toggled.

$$C|10> = C\left(\binom{0}{1} \otimes \binom{1}{0}\right) = C\left(\begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}\right)$$

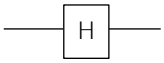
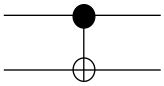
$$= \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}\begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} = \binom{0}{1} \otimes \binom{0}{1} = |11> \quad (5)$$

$$C|11> = C\left(\binom{0}{1} \otimes \binom{0}{1}\right) = C\left(\begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}\right)$$

$$= \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}\begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} = \binom{0}{1} \otimes \binom{1}{0} = |10> \quad (6)$$

Table 2 Quantum Logic Gates, Quantum Network, Truth Table and Equivalent Matrix

Quantum Logic Gate	Symbol	Truth Table		Equivalent Matrix
		Input	**Output**	
X gate	X	|0>	|1>	$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$
		|1>	|0>	
		Input	**Output**	
Z gate	Z	|0>	|0>	$\begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$
		|1>	-|1>	

Quantum Logic Gate	Symbol	Truth Table		Equivalent Matrix
H gate		**Input**	**Output**	$\frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix}$
		|0>	$(\frac{\|0>+\|1>}{\sqrt{2}})$	
		|1>	$(\frac{\|0>-\|1>}{\sqrt{2}})$	
CNOT gate		**Input**	**Output**	$\begin{pmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 0 & 1\\ 0 & 0 & 1 & 0\end{pmatrix}$
		|00>	|00>	
		|01>	|01>	
		|10>	|11>	
		|11>	|10>	

- Case 0: Control qubit is 0. Target qubit won't be changed.

$$C|00> = C\left(\begin{pmatrix}1\\0\end{pmatrix}\otimes\begin{pmatrix}1\\0\end{pmatrix}\right) = C\left(\begin{pmatrix}1\\0\\0\\0\end{pmatrix}\right)$$

$$= \begin{pmatrix}1&0&0&0\\0&1&0&0\\0&0&0&1\\0&0&1&0\end{pmatrix}\begin{pmatrix}1\\0\\0\\0\end{pmatrix} = \begin{pmatrix}1\\0\\0\\0\end{pmatrix} = \begin{pmatrix}1\\0\end{pmatrix}\otimes\begin{pmatrix}1\\0\end{pmatrix} = |00> \quad (7)$$

$$C|01> = C\left(\begin{pmatrix}1\\0\end{pmatrix}\otimes\begin{pmatrix}0\\1\end{pmatrix}\right) = C\left(\begin{pmatrix}0\\0\\0\\1\end{pmatrix}\right)$$

$$= \begin{pmatrix}1&0&0&0\\0&1&0&0\\0&0&0&1\\0&0&1&0\end{pmatrix}\begin{pmatrix}0\\1\\0\\0\end{pmatrix} = \begin{pmatrix}0\\1\\0\\0\end{pmatrix} = \begin{pmatrix}1\\0\end{pmatrix}\otimes\begin{pmatrix}0\\1\end{pmatrix} = |01> \quad (8)$$

1.2.3 X gate and its properties

The X gate operates on a single qubit. It rotates the state of qubit by 180° (π) around the x-axis. Its property involves toggling the qubit's value (|0> => |1> and |1> => |0>). From row 1 column 4 of Table 2, the X gate has a matrix form represented as $\begin{pmatrix}0 & 1\\ 1 & 0\end{pmatrix}$.

If input is qubit |0>, $\begin{pmatrix}1\\0\end{pmatrix}$.

$$X(|0>) = \begin{pmatrix}0 & 1\\ 1 & 0\end{pmatrix}\begin{pmatrix}1\\0\end{pmatrix} = \begin{pmatrix}0\\1\end{pmatrix} = |1> \quad (9)$$

If input is qubit |1>, $\begin{pmatrix}0\\1\end{pmatrix}$.

$$X(|1>) = \begin{pmatrix}0 & 1\\ 1 & 0\end{pmatrix}\begin{pmatrix}0\\1\end{pmatrix} = \begin{pmatrix}1\\0\end{pmatrix} = |0> \quad (10)$$

1.2.4 Z gate and its properties

The Z gate operates on a single qubit. It rotates the state of qubit by 180° (π) around the z-axis, effectively changing the value of the qubit. From row 2 column 4 of Table 2, the Z gate has a matrix form represented as $\begin{pmatrix}1 & 0\\ 0 & -1\end{pmatrix}$.

If input is qubit |0>, $\begin{pmatrix}1\\0\end{pmatrix}$.

$$Z(|0>) = \begin{pmatrix}1 & 0\\ 0 & -1\end{pmatrix}\begin{pmatrix}1\\0\end{pmatrix} = \begin{pmatrix}0\\0\end{pmatrix} = |0> \quad (11)$$

If input is qubit |1>, $\begin{pmatrix}0\\1\end{pmatrix}$.

$$Z(|1>) = \begin{pmatrix}1 & 0\\ 0 & -1\end{pmatrix}\begin{pmatrix}0\\1\end{pmatrix} = \begin{pmatrix}0\\-1\end{pmatrix} = -1\begin{pmatrix}0\\1\end{pmatrix} = -1(|1>) = -|1> \quad (12)$$

2. Quantum Teleportation Overview and Related Mathematics

2.1 Big Picture of Quantum Teleportation

Quantum teleportation utilizes the principles of quantum physics to transfer information from one place to another without the physical movement of particles. However, it's important to note that quantum teleportation is distinct from the teleportation portrayed in science fiction movies, where objects or people can be moved from one place to another.

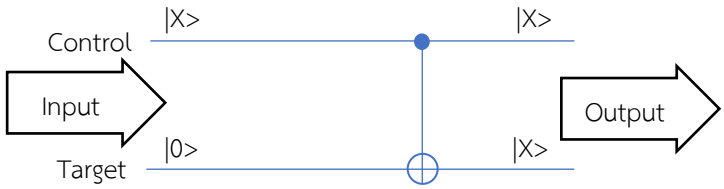


Figure 3 CNOT properties

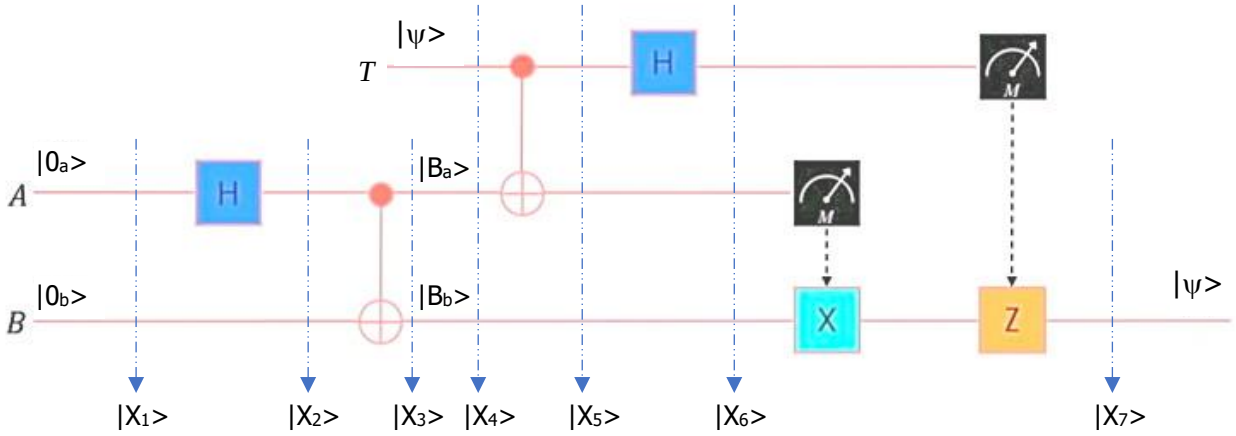


Figure 4 Quantum Teleportation Diagram

Quantum teleportation utilizes quantum entanglement as its fundamental framework[4]. Figure 4 provides a concise diagram illustrating how it operates. The conceptual steps of quantum teleportation are as follows: ψ

1. Initialization: The sender (A) and the receiver (B) share an entangled pair of particles. Each particle in this pair is in a state of superposition (after the H gate) and is entangled with its counterpart. The initial pair consists of |0> for A and |0> for B, denoted further in the calculation as $|0_a\rangle$ and $|0_b\rangle$ respectively. The second pair comprises $T|\psi\rangle$ (the message to be teleported) and $|0_a\rangle$ from A. This means that, regardless of the distance between them, measuring the state of one particle instantaneously determines the state of the other in its pair.

2. Entanglement Measurement: The sender measures both the unknown quantum state to be teleported and their own particle from the entangled pair. After measurement, the unknown state is destroyed.

3. Information Transmission: The sender conveys the measurement result to the receiver through a classical communication channel. This step involves transmitting two classical bits of information. Using a traditional communication channel, the speed of sending two classical bits is as usual, not faster than the speed of light.

4. State Reconstruction: Upon receiving the sender's classical information, the receiver performs suitable quantum operations on their entangled particle based on the measurement result.

5. State Transfer: The entangled particle of the receiver assumes the same state as the original unknown quantum state held by the sender. The unknown quantum state has now been "teleported" from the sender to the recipient.

It's crucial to emphasize that the actual particles containing the quantum state are not physically moved during this process. Instead, the quantum state is effectively transferred from one location to another by leveraging the entanglement between the particles of the sender and receiver.

2.2 Quantum Teleportation's Related Basic Mathematics

Please note that the subscriptions 'a' in $|0_a\rangle$ and 'b' in $|0_b\rangle$ used in all explanations refer to the sender (A) and receiver (B) respectively. However, they can be omitted without any adverse effects. The explanations of the steps in Figure 4 are as follows:.

1. At the starting state, $|X_1\rangle$, there are 2 qubits from A and B. The result is from the tensor product of both qubits.

$$|X_1\rangle = |0_a\rangle \otimes |0_b\rangle = |0_a\rangle|0_b\rangle \quad (13)$$

2. At $|X_2\rangle$ state, $|0_a\rangle$ passes Hadamard gate.

$$|X_2\rangle = H(|0_a\rangle)|0_b\rangle = \frac{|0a\rangle + |1a\rangle}{\sqrt{2}}.|0\rangle_b = \frac{|0a0b\rangle + |1a0b\rangle}{\sqrt{2}} \quad (14)$$

3. At $|X_3\rangle$ state, $|0_a\rangle$ (control qubit) and $|0_b\rangle$ (target qubit) pass CNOT gate.

$$|X_3\rangle = \frac{|0a0b\rangle + |1a1b\rangle}{\sqrt{2}} = |B_{0a0b}\rangle = |B_a\rangle = |B_b\rangle \quad (15)$$

($|B_{0a0b}\rangle$ in (15) is Bell state of 0_a0_b which will be used further for both sender A and receiver B because of the result from CNOT properties will be the same as shown in Figure 3.)

4. At $|X_4\rangle$ state, there are 2 qubits which are $|\psi\rangle$ from T and $|B_a\rangle$ which will be used by sender A.

$$|X_4\rangle = |\psi\rangle \otimes |B_a\rangle = |\psi\rangle|B_a\rangle = |\psi\rangle|B_{0a0b}\rangle$$

As $|\psi\rangle = \alpha|0_a\rangle + \beta|1_a\rangle$ (where $||\alpha||^2 + ||\beta||^2 = 1$), so, $|X_4\rangle$ can be written like this.

$$|X_4\rangle = (\alpha|0_a\rangle + \beta|1_a\rangle)\left(\frac{|0a0b\rangle + |1a1b\rangle}{\sqrt{2}}\right)$$

$$|X_4\rangle = \frac{\alpha|0a0a0b\rangle + \alpha\{0a1a1b) + \beta(1a0a0b) + \beta(1a1a1b)}{\sqrt{2}} \quad (16)$$

5. At $|X_5\rangle$ state, it passes CNOT gate. The result will be as follows.

$$|X_5\rangle = \frac{\alpha|0a0a0b\rangle + \alpha\{0a1a1b) + \beta(1a1a0b) + \beta(1a0a1b)}{\sqrt{2}} \quad (17)$$

6. At $|X_6\rangle$ state, the data goes through H gate. The values of H gate are from row 7 column 4 of Table 2. The processes will be as follows.

$$|X_6\rangle = \frac{1}{\sqrt{2}}\begin{pmatrix} \alpha\left(\frac{|0a\rangle+|1a\rangle}{\sqrt{2}}\right)\left|0a0b\right\rangle + \alpha\left(\frac{|0a\rangle+|1a\rangle}{\sqrt{2}}\right)\left|1a1b\right\rangle + \\ \beta\left(\frac{|0a\rangle-|1a\rangle}{\sqrt{2}}\right)\left|1a0b\right\rangle + \beta\left(\frac{|0a\rangle-|1a\rangle}{\sqrt{2}}\right)\left|0a1b\right\rangle \end{pmatrix}$$

$$|X_6\rangle = \frac{1}{2}\begin{pmatrix} \alpha|0a0a0b\rangle + \alpha|1a0a0b\rangle + \alpha|0a1a1b\rangle + \alpha|1a1a1b\rangle + \\ \beta|0a1a0b\rangle - \beta|1a1a0b\rangle + \beta|0a0a1b\rangle - \beta|1a0a1b\rangle \end{pmatrix}$$

$$|X_6\rangle = \frac{1}{2}\begin{pmatrix} |0a0a\rangle(\alpha|0b\rangle + \beta|1b\rangle) + |1a0a\rangle(\alpha|0b\rangle - \beta|1b\rangle) + \\ |0a1a\rangle(\alpha|1b\rangle + \beta|0b\rangle) + |1a1a\rangle(\alpha|1b\rangle - \beta|0b\rangle) \end{pmatrix} \quad (18)$$

7. At $|X_7\rangle$ state, after sender, A, measures her 2 qubits, she sends these 2 classical bits to the receiver, B. The results of measurement can be either 00, 01, 10, or 11. The actions that the receiver, B, should take after obtaining the measurement result are related to the X and Z gates, as summarized in Table 3. The properties of the X and Z gates are detailed in rows 2 and 4 of Table 2.

Table 3 Information that Receiver Gets and Related Gate(s) for Modifying Result State to Obtain Final Results

Measurement Result	Probability	Result State	Operation and Result
00	1/4	$\alpha\|0_b\rangle + \beta\|1_b\rangle$	No additional operation
10	1/4	$\alpha\|0_b\rangle - \beta\|1_b\rangle$	$Z(\|B_b\rangle) = \alpha\|0_b\rangle + \beta\|1_b\rangle$
01	1/4	$\alpha\|1_b\rangle + \beta\|0_b\rangle$	$X(\|B_b\rangle) = \alpha\|0_b\rangle + \beta\|1_b\rangle$
11	1/4	$\alpha\|1_b\rangle - \beta\|0_b\rangle$	$X(\|B_b\rangle) = \alpha\|0_b\rangle - \beta\|1_b\rangle$ $Z(X(B_b)) = \alpha\|0_b\rangle + \beta\|1_b\rangle$

After performing the operation(s) based on the measurement results, the final output obtained by the receiver, B, will be the same as what the sender, A, sent: $|\psi\rangle = \alpha|0_a\rangle + \beta|1_a\rangle$.

3. Experiments

3.1 Coding for Quantum Teleportation

To develop a practical demonstration of how quantum teleportation works in this research, the following steps are required. First, prepare the development environment based on .NET architectures. Second, write code for Q# library development (a file with a '.dll' extension) that executes quantum teleportation processes on the Microsoft quantum simulator. And third, create a C# client application that connects to the Q# library in the second portion. Figures 6, 7, and 8 display flowcharts of the quantum teleportation application library.

3.1.1 Development Environment Preparation

There are 3 software (or newer) used in this task.

- .NET SDK (version 6.0)
  (https://dotnet.microsoft.com/en-us/download)
- Visual Studio Code (as IDE)
  (https://code.visualstudio.com/download)
- Microsoft Quantum Development Kit for Visual Studio Code (Visual Studio Code Extension)
  (https://marketplace.visualstudio.com/items?itemName=zetta.qsharp-extensionpack)

All above tools can be used under various operating systems that support .NET such as MS Windows, macOS and Linux.

3.1.2 Overall Picture of the Application

The program consists of two parts, as illustrated in Figure 5. The first part is the client, which is written in C#. The second part is the library, written in Q# and, in this project, runs on the quantum simulator. The client sends a message as a boolean to the library. The library receives the boolean input (considered as a binary classical bit) and initializes a qubit appropriate for the inputted value. The teleportation process occurs within the library, and the boolean result is sent back to the client to indicate whether the teleportation process succeeded or not. The working process of this solution resembles a client-server system.

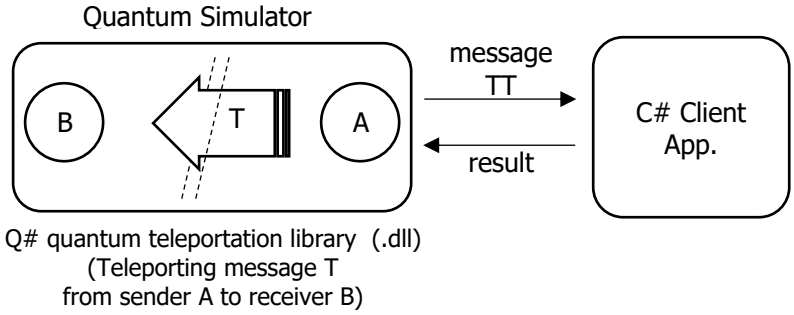


Figure 3 Quantum Teleportation Library and the Interaction with Client

3.1.3 Flowchart for Quantum Teleportation Library

Figure 6 illustrates the flowchart of the core program in the library. The library receives classical bit (boolean) data, 'tt,' from the client program. Two qubits, 't' representing the message T and 'b' representing the receiver B in Figure 4, are prepared and initialized. According to the flowchart, if the initial classical bit message ('tt') is true, the X gate is used to toggle the value of the message qubit 't' to |1>; otherwise, it remains as |0>. The message qubit 't' and the receiver qubit 'b' are then passed as arguments to the 'Teleport' function. After the 'Teleport' function completes its operation, the message qubit 't' is teleported to the receiver qubit 'b'. Subsequently, the receiver qubit 'b' is measured by the 'MeasureReceiver' function, which yields the classical bit result 'bb.' If the value of 'bb' (the message received by the receiver) is the same as 'tt' (the initial message), the library will notify the user on the screen that the teleportation has been successfully completed; otherwise, it is considered a failure.

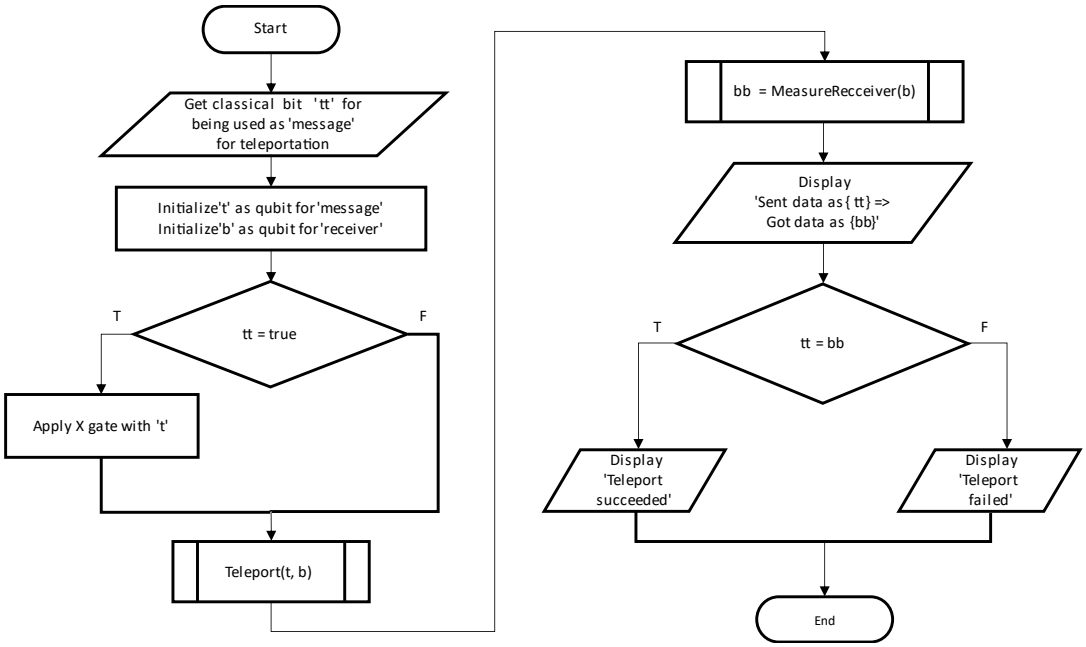


Figure 4 Flowchart of Core Program in the Quantum Teleportation Library

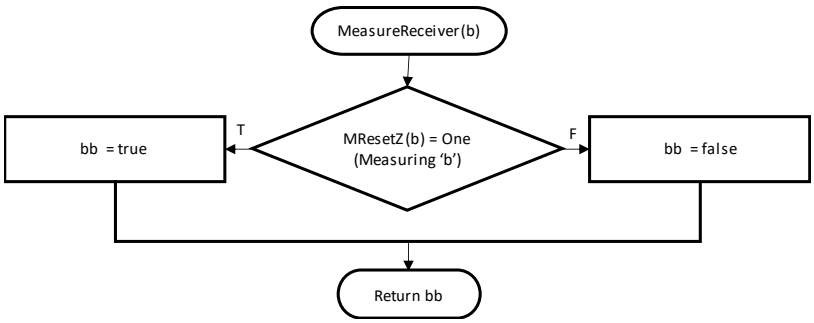


Figure 7 Function "MeasureReceiver"

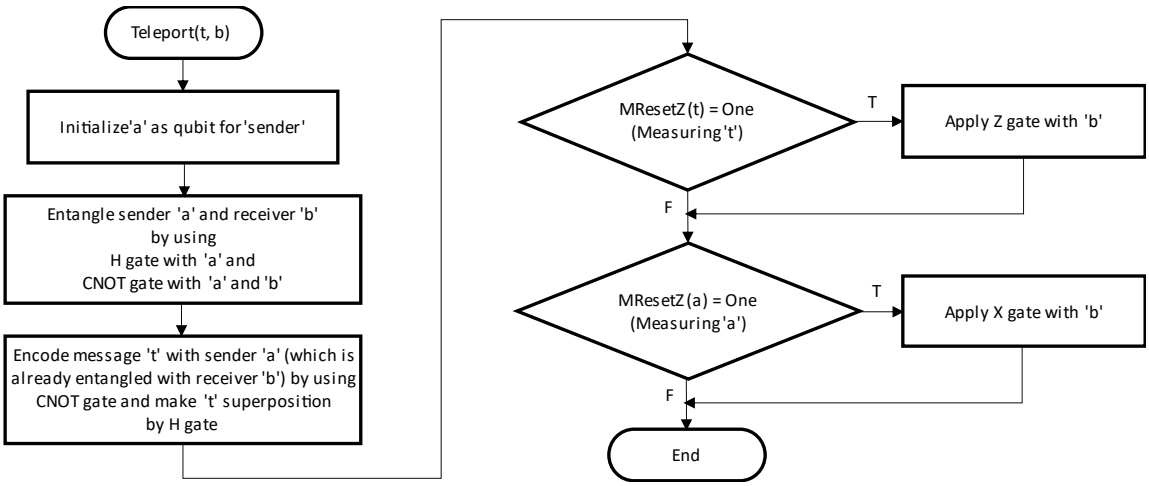


Figure 8 Function "Teleport"

Listing 1 Q# Code of Quantum Teleportation Library (Library.qs)

```
namespace QuantumTeleportation02 {
    open Microsoft.Quantum.Canon;
    open Microsoft.Quantum.Intrinsic;
    open Microsoft.Quantum.Random;
    open Microsoft.Quantum.Measurement;
    // as in Figure 4, a is sender, b is receiver, t is message
    operation Teleport (t : Qubit, b : Qubit) : Unit {
        // initial state of a is |0> (by default)
        use a = Qubit();
        // from Figure 4, entangle sender (a) and receiver (b) by using H and CNOT gates
        H(a);
        CNOT(a, b);
        // from Figure 4, encode the message (t) with sender (a)
        // which is already entangled with receiver (b)
        CNOT(t, a);
        H(t);
        // from Figure 4, message qubit (t), after measured, will be handled by Z gate
        if(MResetZ(t) == One) { Z(b); }
        // from Figure 4, qubit from sender (a), after measured, will be handled by X gate
        if(MResetZ(a) == One) { X(b); }
    }
    // 'tt' is classical bit for message to be transported
    // 'bb' is what receiver gets in form of classical bit (after being measured)
    operation SendData2Teleport (tt : Bool) : Bool {
        // initial state of t is |0> (by default)
        use t = Qubit();
        // initial state of b is |0> (by default)
        use b = Qubit();
        // if 'tt' is true (classical bit is 1), use X gate to toggle initial state of 't' to be |1>
        // else ('tt' is not true (classical bit is 0)) leave 't' to be |0> which is its initial state
        if (tt == true) {
            X(t);
        }
        // call Teleport method, send t (message) to b (receiver)
        Teleport(t, b);
        // measuring received message (and getting as classical bit)
        let bb = MeasureReceiver(b);
        Message($"Sent data as: {tt} => Got data as: {bb}");
        if(tt == bb) {
            Message("Teleportation successful this time.");
            return true;
        } else {
            Message("Teleportation failed this time.");
            return false;
        }
    }
    operation MeasureReceiver(b : Qubit) : Bool {
        // measuring b (receiver)
        // if result is One, measured value is true (1 in classical bit)
        // if result is not One, measured value is false (0 in classical bit)
        // bb is receiver in classical bit
        mutable bb = false;
        if (MResetZ(b) == One) {
            set bb = true;
        } else {
            set bb = false;
        }
        return bb;
    }
}
```

Listing 2 C# Code of Client Program (Program.cs)

```
using System.Globalization;
using Microsoft.Quantum.Simulation.Simulators;
using QuantumTeleportation02;
namespace QuantumTeleportation02Use;
class Program {
   static async Task Main() {
      bool blnResult = false;
      int intRound = 0;
      try {
         using var qsim = new QuantumSimulator();
         Console.WriteLine("Calling Teleportation Sample from Q# Library to C# Code:");
         Console.Write("How many round you want to try (positive integer)? ");
         if(int.TryParse(Console.ReadLine(), NumberStyles.Any, null, out intRound) == true) {
            Random r = new Random();
            for (int i = 0; i < intRound; i++) {
               bool blnRandom = (r.Next(2) == 1);
               Console.WriteLine("Sending {0}.", blnRandom.ToString());
               blnResult = await SendData2Teleport.Run(qsim, blnRandom);
               Console.WriteLine("Report to the caller: teleport # {0} was {1}.", i+1, blnResult == true ? "'successfully'" :
"'failed'");
            }
         }
      } catch(Exception ex) {
         Console.WriteLine("Exception happen: " + ex.Message);
         Console.WriteLine("Explanation: " + ex.StackTrace);
      }
   }
}
```

Figure 7 depicts the flowchart of the 'MeasuredReceiver' function. This function performs quantum measurement on the receiver qubit 'b'. If the result is One, the variable 'bb', used here as a classical bit (boolean) result value for the receiver, is set to 'true'; otherwise, it is set to 'false'. The function then returns 'bb' (a boolean or classical bit value) to its caller.

Figure 8 shows the flowchart of the 'Teleport' function, which receives two data inputs: 't', representing the message qubit, and 'b', representing the receiver qubit. To comprehend the flowchart and code, reference to Figure 4 diagram is necessary. The function begins with the initialization of sender qubit 'a'. Subsequently, both sender qubit 'a' and receiver qubit 'b' are entangled using the H gate on 'a' and the CNOT gate on 'a' and 'b'. Following this, the message qubit 't' and sender qubit 'a' (already entangled with receiver qubit 'b') are encoded using the CNOT gate. Afterwards, the resulting message qubit, 't', passes through the H gate, placing it in a superposition state.

The next two steps involve quantum measurements, as depicted in Figure 4. First, the message qubit 't' is measured. If the result is One, the Z gate is applied to the receiver qubit 'b'; otherwise, no operation is performed. Second, the sender qubit 'a' is also measured. If the result is One, the X gate is applied to the receiver qubit 'b'; otherwise, no operation is performed. At the conclusion of the 'Teleport' function, theoretically, the receiver qubit 'b' is in a state identical to the initial message qubit 't', indicating the successful completion of the message transportation process.

3.1.4 Q# Code for Quantum Teleportation Library

Listing 1 presents the Q# code from the 'Library.qs' file, corresponding to the flowcharts discussed in section 3.1.3. After compilation, this library is saved as 'QuantumTeleportation02.dll'. The library comprises three functions: 'Teleport', 'SendData2Teleport', and 'MeasureReceiver'.

1) First, 'Teleport' takes two arguments as input: the message qubit 't' and the receiver qubit 'b'. This function does not return any value to the caller.
2) Second, 'SendData2Teleport' requires one argument as input: the message in classical bit (boolean) form, denoted as 'tt'. It returns a boolean value to the caller.
3) The last function, 'MeasureReceiver', takes one input argument: the receiver qubit 'b'. It returns a boolean value to its caller.

3.1.5 C# Code for Client Program

Listing 2 provides the C# code from 'Program.cs,' which serves as the client program to initiate work with the teleportation library discussed in section 3.1.4. To use the client program, the user needs to place the compiled library file, 'QuantumTeleportation02.dll,' in the same folder where 'Program.cs' is located. The steps involved in the client code are as follows:

1) Firstly, the user is prompted to enter the number of rounds for iterating the sending of binary classical bit messages to the library.
2) Next, a random boolean value is generated and sent to the library via the quantum simulator. Working with the teleportation library in this case employs asynchronous methods.
3) The result of the teleportation process is returned from the library.
4) The second and third steps are iterated until the specified number of rounds entered in the first step is reached.Results

```
PS D:\workroom\LearnQuantumVSCode\QuantumTeleportation02Use> dotnet run
Calling Teleportation Sample from Q# Library to C# Code:
How many round you want to try (positive integer)? 5
Sending True.
Sent data as: True => Got data as: True
Teleportation successful this time.
Report to the caller: teleport # 1 was 'successfully'.
Sending True.
Sent data as: True => Got data as: True
Teleportation successful this time.
Report to the caller: teleport # 2 was 'successfully'.
Sending False.
Sent data as: False => Got data as: False
Teleportation successful this time.
Report to the caller: teleport # 3 was 'successfully'.
Sending True.
Sent data as: True => Got data as: True
Teleportation successful this time.
Report to the caller: teleport # 4 was 'successfully'.
Sending False.
Sent data as: False => Got data as: False
Teleportation successful this time.
Report to the caller: teleport # 5 was 'successfully'.
PS D:\workroom\LearnQuantumVSCode\QuantumTeleportation02Use>
```

Figure 5 Result of Running

The sample results of the program are illustrated in Figure 9. The client program prompts the user to enter the number of rounds for which they want random boolean values to be sent as input. Each round begins with a boolean classical bit value that is converted into a qubit and transmitted to the quantum simulator to initiate the teleportation process. Once the teleportation is completed, the library in the simulator displays the following results: the value of the qubit that was sent, the value of the qubit received from the teleportation process, and the status indicating whether the teleportation was successful or not. These processes are executed iteratively until the specified number of rounds is completed. The program effectively demonstrates that the original message sent by the sender can reach the receiver through teleportation processes, as per the theoretical framework.

4. Conclusions

Teleportation is one of the renowned fundamental principles in quantum computing. It is built upon quantum entanglement, which has been previously explained and proven in prior researches [7][8]. This paper elucidates quantum teleportation by utilizing basic mathematics to demonstrate how it effectively transports message qubits from the sender to the receiver. To simplify the deployment and comprehension of quantum computing in real-world applications, the concept of quantum logic gates is employed, replacing complex mathematical processes. The current research demonstrates the step-by-step creation of runnable teleportation programs in Q#, as outlined in theory, culminating in the development of a library software (.dll file) suitable for various applications. Samples illustrating the use of teleportation functions from the compiled Q# library in a C# client program are also provided. This practical research offers significant benefits to quantum computing scholars, particularly newcomers, by providing clear explanations through simple mathematical concepts and offering real, runnable code samples for their further research.

5. Future Works

After clarifying the steps involved in creating the teleportation library and confirming its correct functionality with a console application, future developments could focus on enhancing the solution into a WebAPI structure. This enhancement would enable the library to facilitate two-way communication with various client platforms, including Windows, web, mobile, and even IoT applications

References

[1] B. Marr, "*Quantum Computing Now And In The Future: Explanation, Applications, And Problems*," [Online]. Available: http://Forbes, Aug. 26, 2022. https://www.forbes.com/sites/bernardmarr/2022/08/26/quantum-computing-now-and-in-the-future-explanation-applications-and-problems/?sh=38224fed1a6b. [Accessed: Mar. 14, 2023].

[2] J. Keane, "*The race toward a new computing technology is heating up — and Asia is jumping on the trend*," [Online]. Available: http://www.cnbc.com/2022/06/07/quantum-computing-more-asian-countries-are-getting-in-on-the-trend.html. [Accessed: Mar. 14, 2023].

[3] V. Sankaran, "*China builds world's fastest programmable quantum computers that outperform 'classical' computers*," [Online]. Available: https://www.independent.co.uk/tech/china-scientists-programmable-quantum-computers-b1946018.html. [Accessed: Apr. 26, 2023].

[4] N. D. Mermin, *Quantum Computer Science: An Introduction*, Cambridge University Press, 2007.

[5] A. Bellapu, "*10 Difficult Problems Quantum Computers can Solve Easily*," [Online]. Available: https://www.analyticsinsight.net/10-difficult-problems-quantum-computers-can-solve-easily. [Accessed: June 06, 2023].

[6] A. Tatourian, "*Lecture Notes of Quantum Computing for Computer Scientists*," [Online]. Available: https://tatourian.blog/2018/09/01/quantum-computing-for-computer-scientists. [Accessed: May 01, 2023].

[7] K. Chantarasathaporn, C. Thaiupathump, et al., "Practical Entanglement for Quantum Computing on Quantum Simulator by Q#," *The 19th International Conference in Applied Computer Technology and Information System*, Bangkok: Southeast Bangkok University, Mar. 2023, pp. 408–418.

[8] K. Chantarasathaporn, C. Thaiupathump, et al., "Web API and Quantum Simulator for Testing Quantum Entanglement Concepts," *International Journal of Applied Computer Technology and Information Systems*, vol. 13, no. 1, pp.17-24, April 2023 - September 2023.

[9] A. S. Tolba, M. Z. Rashad, M. A. El-Dosuky, "*Q#, a quantum computation package for the .NET platform*," [Online]. Available: https://arxiv.org/abs/1302.5133. [Accessed: May 01, 2023].

[10] Microsoft DevLabs, "*Microsoft Quantum Development Kit for Visual Studio Code*," [Online]. Available: https://marketplace.visualstudio.com/items?itemName=quantum.quantum-devkit-vscode. [Accessed: May 01, 2023].

[11] Z. Hussain and A. Talib, "Strengths and Weaknesses of Quantum Computing," *International Journal of Scientific & Engineering Research*, vol. 7, no. 9, pp.1526-1531, September 2016.

[12] D. Voorhoede, "*Hadamard gate*," [Online]. Available: https://www.quantum-inspire.com/kbase/hadamard. [Accessed: June 10, 2023].

[13] H. Paudel, M. Syamlal, and S. Crawford, "Quantum Computing and Simulations for Energy Applications: Review and Perspective," *Open Access J. Am. Chem. Soc.*, vol. 2, no. 3, pp. 151–196, January 2022.