

Web API and Quantum Simulator for Testing Quantum Entanglement Concepts

Kayun Chantarasathaporn¹, Choonhapong Thaiupathump², Nirun Ngamkerd³,
Poramate Ruksawong⁴, Suphanut Kathong⁵ and Sudasawan Ngammongkolwong⁶

^{1,2,3,4,5}Department of Information Technology and Management,
Faculty of Business Administration, Krirk University, Bangkok, Thailand
e-mail: ¹kayun.cha@krirk.ac.th, ²choonhapong.tha@staff.krirk.ac.th
³nirun.nga@staff.krirk.ac.th, ⁴pruksawong@gmail.com, ⁵supanut@saijai-tech.com
⁶Faculty of Science and Technology
Southeast Bangkok University, Bangkok, Thailand
e-mail: sudasawan@southeast.ac.th

Abstract—Quantum computing uses concept of quantum physics to deploy on next generation quantum computers. Several inventions and innovations in this field in the last two decades can push the theory to real works much faster than what in the past. This paper is intended for students who are interested in quantum software development and related mathematical concepts. Simple matrix explanations of fundamental quantum entanglement made it easier to understand. Showing how to create Q# quantum library (.dll) that operates on quantum simulator, links with C# Web API and works with Swagger web client makes interested learners clearer about how to generate program from supporting theories and tools. Running results of this sample development were compatible with the mathematical principle of quantum entanglement and fit that concept. This paper can be a good kickstart for learners who want to clarify concepts and really create basic quantum programs on his or her own computer.

Keywords: *quantum computing; quantum programming; Q#; QSharp; entanglement; quantum simulator; Web API*

I. INTRODUCTION

The concept of quantum mechanics in modern physics has been widespread rapidly in IT world during recent two decades due to the joining and contribution of major vendors such as Microsoft, IBM, Google and Amazon[1]. More sophisticated tools, hardware and software have also been shared for public use at low or no cost. In part of learning quantum computing concepts, many more simplified versions of explanation have been released. From all supporting reasons, more and more people are persuaded to try and learn quantum computing.

Classical computing uses binary digits, or bits, whereas quantum computing uses quantum bits, or qubits. Unlike classical computing, which employs bits with values of 0 or 1, quantum computing uses qubits with values of both 0 and 1. Superposition is the idea that allows a qubit value to be both 0 and 1 at the same time. Aside from superposition, other well-known fundamental concepts in quantum computing that should be learned include entanglement and teleportation.

Quantum computing research has been handled in both western and eastern countries. In Asian side, China, Japan and India have been considered the leaders[2]. University of Science and Technology of China (USTC) announced in 2021 that they could invent quantum computer that the operation was based on light which was much faster than the former technology that the westerns used[3].

Interest of quantum computing can be divided into parts of hardware and software[4]. Hardware portion can also be classified to physical hardware (standalone and cloud base) and hardware simulator. Software chunk can be divided into system software (such as operating system)[5] and application software[6].

This paper targets in providing simple explanation of quantum entanglement and showing how to create related software. Figure 1 illustrates the software architecture. Quantum entanglement library (.dll), that operates on quantum simulator, is written in Q#. This library is consumed by client which is built in form of .NET Web API written in C#. As a Web API, any platform of client that can communicate with it, such as, web, windows, mobile or even IoT apps can then communicate back and forth with quantum entanglement library that operates on quantum simulator as shown in Figure 1.

Contents of this article are as follows: a brief discussion of bits and qubits, quantum gates and related mathematics, an overview of quantum entanglement, a demonstration of building a Q# library, a sample creation of C# Web API, Web API consumption, and the prove that the created software solution could provide results that corresponded to the theory. As this development was run on quantum simulator, learners will learn how to use their own standard computers to practice quantum programming which help sharpen their skill to be able to learn more complicated quantum computing topics.

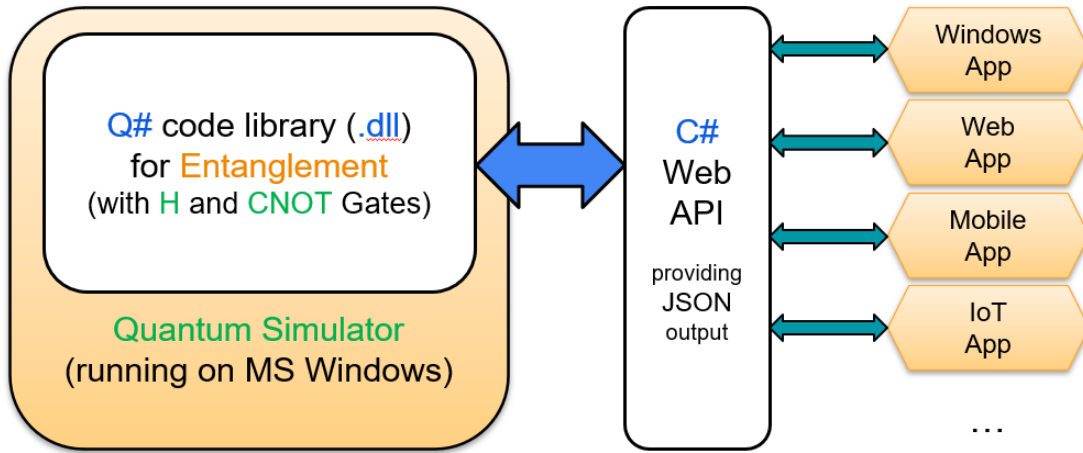


Figure 1. Various kinds of app can communicate through Web API to quantum entanglement code library.

II. FUNDAMENTAL OF QUANTUM COMPUTING

A. Bit and Qubit

A classical bit is the fundamental unit of classical computing and can only have one of two values: 0 or 1. A qubit, on the other hand, is the fundamental unit of quantum computing and may be both 0 (Dirac notation for 0 is $|0\rangle$) and 1 (Dirac notation for 1 is $|1\rangle$) at the same time. This is referred to as superposition state. It can be written as $\frac{|0\rangle + |1\rangle}{\sqrt{2}}$.

Figure 2. Classical Bit and Qubit[7]

B. Vector Form and Dirac Notation

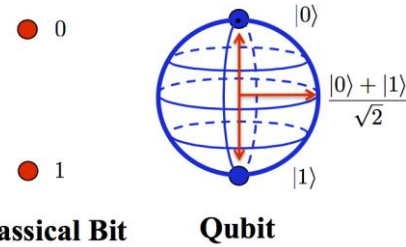
$\begin{pmatrix} 1 \\ 0 \end{pmatrix}$ is a vector form of $|0\rangle$ while $\begin{pmatrix} 0 \\ 1 \end{pmatrix}$ is a vector format of $|1\rangle$. From [4], superposition of one qubit ($\frac{|0\rangle + |1\rangle}{\sqrt{2}}$) can be written in vector form as $\begin{pmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{pmatrix}$. For 2 qubits, they can be written in the format of vector as well. [4] shows that $|00\rangle$ has vector form as $\begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}$ while $|01\rangle$ has $\begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}$ as its vector. $\begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}$ is a vector form of $|10\rangle$ while $|11\rangle$ has $\begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}$ as its vector.

C. Essential Quantum Logic Gates for Quantum Entanglement.

The basic building blocks for creating quantum networks (quantum circuits), which are essential for carrying out quantum computation, are quantum logic gates. They can all be explained in matrix or vector form.

The following are the gates used in creating entanglement.

- H gate or quantum Hadamard gate: The H gate is used to create superpositions of qubits. It takes a qubit in the $|0\rangle$



state to a state that is equally likely to be measured as $|0\rangle$ or $|1\rangle$.

- CNOT gate or quantum controlled-NOT gate: The CNOT gate is a two-qubit gate that flips the second (target) qubit if and only if the first (control) qubit is $|1\rangle$.

Quantum gates can be written or drawn as the following equivalent circuit, matrix, and truth table shown in Table 1.

- H gate

Hadamard gates can convert a qubit from one in usual status to one in status of superposition and vice versa. When a qubit is in superposition, it is thought to be in a probabilistic state, as opposed to its typical deterministic state.

As shown in Table 1, H gate can be written in matrix form as $\begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{pmatrix}$. The results when it operates with qubit $|0\rangle$ and $|1\rangle$ are as shown in (1) and (2).

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

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

For H gate, its product result when operating with superposition status vector will bring back qubits to usual status. This can be proved in (3) and (4).

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

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

State machine diagram of Hadamard gate is shown in Figure 3.

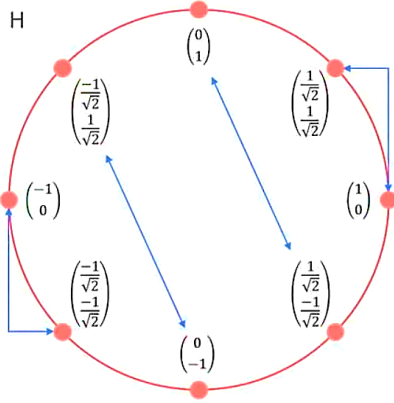


Figure 3. State Machine of H gate [8]

- CNOT gate

CNOT gate works with 2 qubits. While the first qubit is called 'control' qubit, the second qubit is called 'target' qubit. If the control qubit is 1, the target qubit is toggled ($0 \Rightarrow 1$ or $1 \Rightarrow 0$). If the control qubit is 0, the target qubit is unchanged ($0 \Rightarrow 0$ or $1 \Rightarrow 1$). Control qubit itself will never be changed.

$$\begin{array}{ll} 00 \Rightarrow 00 & 10 \Rightarrow 11 \\ 01 \Rightarrow 01 & 11 \Rightarrow 10 \end{array}$$

Matrix form of CNOT gate is as follows.

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

The following shows how CNOT gate can toggle the target qubit when the control qubit is 1. Remember that qubit $|0\rangle$ has vector form as $\begin{pmatrix} 1 \\ 0 \end{pmatrix}$ while qubit $|1\rangle$ has $\begin{pmatrix} 0 \\ 1 \end{pmatrix}$ as its vector form.

$$\begin{aligned} C|10\rangle &= C\left(\begin{pmatrix} 0 \\ 1 \end{pmatrix} \otimes \begin{pmatrix} 1 \\ 0 \end{pmatrix}\right) = C\left(\begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}\right) \\ &= \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \otimes \begin{pmatrix} 0 \\ 1 \end{pmatrix} = |11\rangle \end{aligned}$$

$$\begin{aligned} C|11\rangle &= C\left(\begin{pmatrix} 0 \\ 1 \end{pmatrix} \otimes \begin{pmatrix} 0 \\ 1 \end{pmatrix}\right) = C\left(\begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}\right) \\ &= \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \otimes \begin{pmatrix} 1 \\ 0 \end{pmatrix} = |10\rangle \end{aligned}$$

Next, for CNOT gate, the target qubit will not be changed if the control qubit is 0.

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

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

TABLE 1. GATES, QUANTUM CIRCUIT, MATRIX, AND TRUTH TABLE[9]

| Gates and rotations in Bloch sphere | Circuit representation | Matrix representation | Truth table |
|---|------------------------|--|--|
| <i>I</i> gate: no rotation is performed. | | $I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$ | Input Output 0⟩ 0⟩ 1⟩ 1⟩ |
| <i>X</i> gate: rotates the qubit state by π about the x-axis. | | $X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$ | Input Output 0⟩ 1⟩ 1⟩ 0⟩ |
| <i>Y</i> gate: rotates the qubit state by π about the y-axis. | | $Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}$ | Input Output 0⟩ $i 1\rangle$ 1⟩ $-i 0\rangle$ |
| <i>Z</i> gate: rotates the qubit state by π about the z-axis. | | $Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$ | Input Output 0⟩ 0⟩ 0⟩ $- 1\rangle$ |
| <i>S</i> gate: rotates the qubit state by $\pi/2$ about the z-axis. | | $S = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\pi/2} \end{pmatrix}$ | Input Output 0⟩ 0⟩ 1⟩ $e^{i\pi/2} 1\rangle$ |
| <i>T</i> gate: rotates the qubit state by $\pi/4$ about the z-axis. | | $T = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{pmatrix}$ | Input Output 0⟩ 0⟩ 1⟩ $e^{i\pi/4} 1\rangle$ |
| <i>H</i> gate: rotates the qubit state by π about an axis diagonal in the x-z plane. This is equivalent to an X-gate followed by a $\pi/2$ rotation about the y-axis. | | $H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$ | Input Output 0⟩ $\frac{ 0\rangle + 1\rangle}{\sqrt{2}}$ 1⟩ $\frac{ 0\rangle - 1\rangle}{\sqrt{2}}$ |
| CNOT gate: applies an X-gate to the target qubit if the control qubit is in state 1⟩. | | $CNOT = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$ | Input Output 00⟩ 00⟩ 01⟩ 01⟩ 10⟩ 11⟩ 11⟩ 10⟩ |
| CPHASE gate: apply a Z-gate to the target qubit if the control qubit is in state 1⟩. | | $CPHASE = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix}$ | Input Output 00⟩ 00⟩ 01⟩ 01⟩ 10⟩ 10⟩ 11⟩ $- 11\rangle$ |

III. QUANTUM ENTANGLEMENT IN BRIEF

Mathematics behind quantum computing can be explained easier when using vector and matrix. Quantum

gates can also help explain quantum entanglement in simple matrix form.

A. Quantum Entanglement and Matrix

If the product state of 2 qubits cannot be factored, these 2 qubits are considered entangled.

Tensor product of $\begin{pmatrix} a \\ b \end{pmatrix} \otimes \begin{pmatrix} c \\ d \end{pmatrix}$ is $\begin{pmatrix} ac \\ ad \\ bc \\ bd \end{pmatrix}$, so, it can be

factored out to get the values of ac, ad, bc and bd. If we try

to factor out the vector $\begin{pmatrix} \frac{1}{\sqrt{2}} \\ 0 \\ 0 \\ \frac{1}{\sqrt{2}} \end{pmatrix}$, the results should be as follows.

$$1) ac = \frac{1}{\sqrt{2}} \quad 2) ad = 0 \quad 3) bc = 0 \quad 4) bd = \frac{1}{\sqrt{2}}$$

However, when seeing carefully, it can be seen that this vector is unable to factor out because if $ad = 0$, it means either a or d or both are 0. If a is 0, ac should also be 0, but it is not. In other way, if d is 0, bd should be 0, but it is not. When looking at bc, if $bc = 0$, it means either b or c or both are 0. If b is 0, bd should also be 0, but it is not. Similarly, if c is 0, ac should also be 0, but it is not.

Therefore, the above quantum state cannot be factored out. Collapsing chance to $|00\rangle$ is 0.5 (50%) while collapsing chance to $|11\rangle$ is also 0.5 (50%). In another way, chances of collapsing to $|01\rangle$ or $|10\rangle$ are both 0 (0%).

B. Quantum Entanglement and Quantum Gates

From the above demonstration, vector $\begin{pmatrix} \frac{1}{\sqrt{2}} \\ 0 \\ 0 \\ \frac{1}{\sqrt{2}} \end{pmatrix}$ is in

entangled state. For simplicity, in quantum computing, it is more often to use quantum gates than complicated equations. Figure 4 shows that using H (Hadamard) and CNOT gates can generate entangled state.

$|0\rangle$

$|0\rangle$

Figure 4. H and CNOT gates can create entangled state

C. How H and CNOT Can Generate Entangled State.

This paragraph will describe how both H and CNOT gates can generate entangled state. To find vector of qubits after using Hadamard gate, H gate's State Machine diagram in Figure 3 is referred.

- Case - control qubit is $|0\rangle$ and target qubit is $|0\rangle$:

$$C(H_{(0)}^{(1)} \otimes \begin{pmatrix} 1 \\ 0 \end{pmatrix}) = C\left(\begin{pmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{pmatrix} \otimes \begin{pmatrix} 1 \\ 0 \end{pmatrix}\right) = C\begin{pmatrix} \frac{1}{\sqrt{2}} \\ 0 \\ \frac{1}{\sqrt{2}} \\ 0 \end{pmatrix}$$

$$= \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \\ 0 \\ 0 \end{pmatrix}$$

- Case - control qubit is $|0\rangle$ and target qubit is $|1\rangle$:

$$C(H_{(0)}^{(1)} \otimes \begin{pmatrix} 0 \\ 1 \end{pmatrix}) = C\left(\begin{pmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{pmatrix} \otimes \begin{pmatrix} 0 \\ 1 \end{pmatrix}\right) = C\begin{pmatrix} 0 \\ \frac{1}{\sqrt{2}} \\ 0 \\ \frac{1}{\sqrt{2}} \end{pmatrix}$$

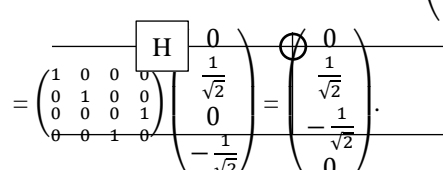
$$= \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 0 \\ \frac{1}{\sqrt{2}} \\ 0 \\ \frac{1}{\sqrt{2}} \end{pmatrix} = \begin{pmatrix} 0 \\ \frac{1}{\sqrt{2}} \\ 0 \\ \frac{1}{\sqrt{2}} \end{pmatrix}$$

- Case - control qubit is $|1\rangle$ and target qubit is $|0\rangle$:

$$C(H_{(1)}^{(0)} \otimes \begin{pmatrix} 1 \\ 0 \end{pmatrix}) = C\left(\begin{pmatrix} \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} \end{pmatrix} \otimes \begin{pmatrix} 1 \\ 0 \end{pmatrix}\right) = C\begin{pmatrix} \frac{1}{\sqrt{2}} \\ 0 \\ -\frac{1}{\sqrt{2}} \\ 0 \end{pmatrix}$$

$$= \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} \frac{1}{\sqrt{2}} \\ 0 \\ -\frac{1}{\sqrt{2}} \\ 0 \end{pmatrix} = \begin{pmatrix} \frac{1}{\sqrt{2}} \\ 0 \\ 0 \\ -\frac{1}{\sqrt{2}} \end{pmatrix}$$

- Case - control qubit is $|1\rangle$ and target qubit is $|1\rangle$:

$$C(H_{(1)}^{(0)} \otimes \begin{pmatrix} 0 \\ 1 \end{pmatrix}) = C\left(\begin{pmatrix} \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} \end{pmatrix} \otimes \begin{pmatrix} 0 \\ 1 \end{pmatrix}\right) = C\begin{pmatrix} 0 \\ \frac{1}{\sqrt{2}} \\ 0 \\ -\frac{1}{\sqrt{2}} \end{pmatrix}$$


$$= \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 0 \\ \frac{1}{\sqrt{2}} \\ 0 \\ -\frac{1}{\sqrt{2}} \end{pmatrix} = \begin{pmatrix} 0 \\ \frac{1}{\sqrt{2}} \\ 0 \\ -\frac{1}{\sqrt{2}} \end{pmatrix}$$

From all entangled cases above, each case's sum of probabilities are as follows respectively.

- Case - control qubit is $|0\rangle$ and target qubit is $|0\rangle$:

$$\begin{matrix} |00\rangle & |01\rangle & |10\rangle & |11\rangle \\ \left\| \frac{1}{\sqrt{2}} \right\|^2 & \|0\|^2 & \|0\|^2 & \left\| \frac{1}{\sqrt{2}} \right\|^2 \\ 0.5 & 0 & 0 & 0.5 \end{matrix} = 1$$

- Case - control qubit is $|0\rangle$ and target qubit is $|1\rangle$:

$$\begin{matrix} |00\rangle & |01\rangle & |10\rangle & |11\rangle \\ \|0\|^2 & \left\| \frac{1}{\sqrt{2}} \right\|^2 & \left\| \frac{1}{\sqrt{2}} \right\|^2 & \|0\|^2 \\ 0 & 0.5 & 0.5 & 0 \end{matrix} = 1$$

- Case - control qubit is $|1\rangle$ and target qubit is $|0\rangle$:

$$\begin{array}{cccc} 0 & 0.5 & 0.5 & 0 \\ |00\rangle & |01\rangle & |10\rangle & |11\rangle \\ \left\| \frac{1}{\sqrt{2}} \right\|^2 & + \|0\|^2 & + \|0\|^2 & + \left\| -\frac{1}{\sqrt{2}} \right\|^2 = 1 \\ 0.5 & 0 & 0 & 0.5 \end{array}$$

- Case - control qubit is $|1\rangle$ and target qubit is $|1\rangle$:

$$\begin{array}{cccc} |00\rangle & |01\rangle & |10\rangle & |11\rangle \\ \|0\|^2 & + \left\| \frac{1}{\sqrt{2}} \right\|^2 & + \left\| -\frac{1}{\sqrt{2}} \right\|^2 & + \|0\|^2 = 1 \\ 0 & 0.5 & 0.5 & 0 \end{array}$$

This can prove that using Hadamard and CNOT gates together can generate entangled state with total chances as expected.

IV. CODING FOR QUANTUM ENTANGLEMENT

To develop solution as shown in Figure 1, the following steps are needed, environment preparation, creating Q# library (.dll) that performs quantum entanglement on quantum simulator, building .NET Web API (C#) that communicates with Q# library. From Web API, any kinds of clients that can consume Web API, such as, windows, web, mobile or even IoT apps are able to connect with quantum system (in this case, it is quantum simulator).

A. Preparation

The following tools are used in this project.

- .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=zeta.qsharp-extensionpack>)

Microsoft Quantum Development Kit for Visual Studio Code can be downloaded from Extension Marketplace. The tools mentioned above can be used under various operating systems that support .NET such as MS Windows, macOS and Linux.

B. Creating Q# Code Library for Quantum Entanglement

The following Q# code is for making quantum entanglement library. The compiled file will be named as QuantumLib02.dll. Source code shows sample usage of H and CNOT gate functions. The program iterates 1,000 times. There are 4 inputs for BellState function, controlInitialState (Bool), targetInitialState (Bool), controlMeasurementBasis (Pauli) and targetMeasurementBasis (Pauli). After measuring, if resultControl is equal to resultTarget, matchingMeasurement will be increased by one.

```
// QuantumLib02\Library.q# -> QuantumLib02.dll
namespace QuantumLib02 {

    open Microsoft.Quantum.Canon;
    open Microsoft.Quantum.Intrinsic;
    open Microsoft.Quantum.Preparation;
    open Microsoft.Quantum.Convert;
```

```
operation BellState(controlInitialState : Bool,
targetInitialState : Bool,
controlMeasurementBasis : Pauli,
targetMeasurementBasis : Pauli) : String {
    mutable matchingMeasurement = 0;
    mutable zeroZero = 0;
    mutable zeroOne = 0;
    mutable oneZero = 0;
    mutable oneOne = 0;
    mutable zz = 0.0;
    mutable zo = 0.0;
    mutable oz = 0.0;
    mutable oo = 0.0;
    mutable allInt = 0;
    mutable allDouble = 0.0;
    for run in 0..999 {
        use (control, target) = (Qubit(), Qubit());
        // prepare |0> or |1> initial state
        PrepareQubitState(control,
controlInitialState);
        PrepareQubitState(target,
targetInitialState);
        H(control); // Hadamard gate
        CNOT(control, target); // CNOT gate
        // PrepareEntangledState([control],
[target]);
        let resultControl =
Measure([controlMeasurementBasis], [control]);
        let resultTarget =
Measure([targetMeasurementBasis], [target]);
        ResetAll([control, target]);
        set zeroZero += resultControl == Zero and
resultTarget == Zero ? 1 | 0;
        set zeroOne += resultControl == Zero and
resultTarget == One ? 1 | 0;
        set oneZero += resultControl == One and
resultTarget == Zero ? 1 | 0;
        set oneOne += resultControl == One and
resultTarget == One ? 1 | 0;
        set matchingMeasurement += resultControl ==
resultTarget ? 1 | 0;
    }
    set allInt = zeroZero + zeroOne + oneZero +
oneOne;
    set allDouble = IntAsDouble(allInt);
    set zz = IntAsDouble(zeroZero) / allDouble;
    set zo = IntAsDouble(zeroOne) / allDouble;
    set oz = IntAsDouble(oneZero) / allDouble;
    set oo = IntAsDouble(oneOne) / allDouble;
    mutable r = ("Initial system state: | " +
controlInitialState ? "1" | "0") +
(targetInitialState ? "1" | "0") + ">" + ("
|00>: " + DoubleAsString(zz)) + (" |01>: " +
DoubleAsString(zo)) + (" |10>: " +
DoubleAsString(oz)) + (" |11>: " +
DoubleAsString(oo)) + (" Measurements of two
qubits matched: " +
IntAsString(matchingMeasurement)) + (" All sum:
" + DoubleAsString(allDouble/allDouble));
    return r;
}

operation PrepareQubitState(qubit : Qubit,
initialState : Bool) : Unit is Adj {
    if (initialState) {
        X(qubit);
    }
}
```

C. Calling Q# Library with C# and Running on Quantum Simulator

To create C# Web API application project in Visual Studio Code environment, the following command needs to

be run in terminal. In this case, the project name is QuantumLib01Use02.

```
dotnet new webapi -o QuantumLib01Use02
```

Make sure that the "NuGet Package Manager" extension is installed in Visual Studio Code because there are required packages needed to be included. Adding packages from NuGet Package Manager can be done via Command Palette.

Command Palette (Ctrl + Shift + P) : NuGet Package Manager : Add Package

The newest version of both Microsoft.Quantum.Simulators and Microsoft.Quantum.QSharp.Core have to be downloaded and installed from NuGet Package Manager.

The following C# code is running and connecting to Q# library. Measuring qubits is done in PauliZ for both control and target qubits. Q# library, QuantumLib02.dll, must be put in the same folder as C# file.

```
// QuantumLib01Use02\Program.cs
using Microsoft.Quantum.Simulation.Core;
using Microsoft.Quantum.Simulation.Simulators;
using QuantumLib02;

var builder =
WebApplication.CreateBuilder(args);
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen(c => {
c.SwaggerDoc("v1", new() { Title="Test API",
Version="v1" });
});
var app = builder.Build();

app.MapGet("/Entanglement", async (int c, int t)
=> {
using var qsim = new QuantumSimulator();
bool blnControl, blnTarget;
blnControl = (c == 0) ? false : true;
blnTarget = (t == 0) ? false : true;
string? result = await BellState.Run(qsim,
blnControl, blnTarget, Pauli.PauliZ,
Pauli.PauliZ);
return Results.Ok(result);
});

app.UseSwagger();
app.UseSwaggerUI(c => {
c.SwaggerEndpoint("/swagger/v1/swagger.json",
"v1");
c.InjectStylesheet("/swagger/custom.css");
c.RoutePrefix = String.Empty;
});

app.Run();
```

D. Modifying Project File to be Able to Work with '.dll'.

The .csproj file needs to be updated as follows to let the project know that it will include 'QuantumLib02.dll' file.

Content of .csproj file after being updated will be as follows. (In this case, getting latest versions of Microsoft.Quantum.Simulators and Microsoft.Quantum.QSharp.Core via NuGet is required.)

```
<!--QuantumLib01Use02\QuantumLib01Use02.csproj-->
<Project Sdk="Microsoft.NET.Sdk">
  <PropertyGroup>
    <OutputType>Exe</OutputType>
    <TargetFramework>net6.0</TargetFramework>
  </PropertyGroup>
  <ItemGroup>
```

```
<PackageReference
Include="Swashbuckle.AspNetCore"
Version="6.5.0"/>
<PackageReference
Include="Microsoft.Quantum.Simulators"
Version="0.28.263081"/>
<PackageReference
Include="Microsoft.Quantum.QSharp.Core"
Version="0.28.263081"/>
</ItemGroup>
<ItemGroup>
  <Reference Include="QuantumLib02">
    <HintPath>QuantumLib02.dll</HintPath>
  </Reference>
</ItemGroup>
</Project>
```

E. Results of the Solution

Web API can work with any platform of client. In this experiment, a well-known API web client named "Swagger" is applied. Usually, plain output from Web API can be in JSON text. Running Web API from the Swagger mask can render the result in easy to check format as shown in Figure 5 and 6.

```
PS D:\workroom\LearnQuantumVSCode\QuantumLib01UseWebAPI02> dotnet run
Building...
info: Microsoft.Hosting.Lifetime[14]
Now listening on: http://localhost:5165
info: Microsoft.Hosting.Lifetime[0]
Application started. Press Ctrl+C to shut down.
info: Microsoft.Hosting.Lifetime[0]
Hosting environment: Development
info: Microsoft.Hosting.Lifetime[0]
Content root path: D:\workroom\LearnQuantumVSCode\QuantumLib01UseWebAPI02
```

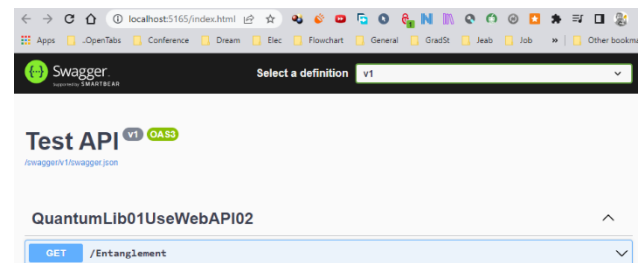


Figure 5. Using Swagger Template to Work with Web API

Figure 6 shows that the input data for entanglement Web API are c (control qubit) and t (target qubit). Values for c and t can be either 0 or 1. Output from this Web API is in string format.

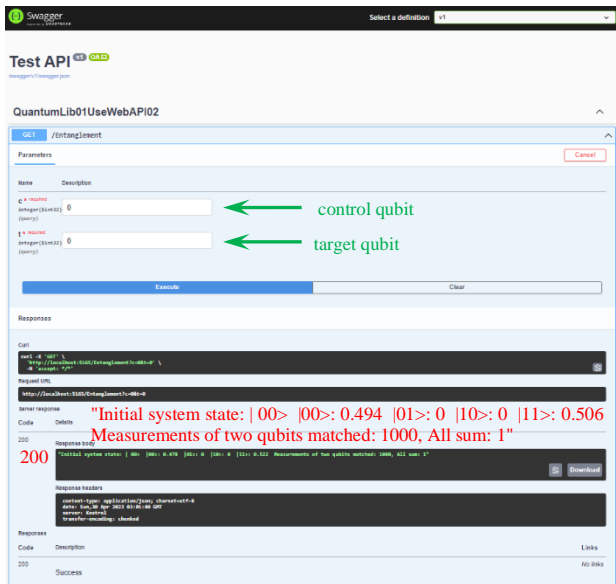


Figure 6. Sending Input and Get Output from Web API

HTTP response value from this entanglement Web API will be 200 (Success) and outputs of all 4 cases are as follows.

- $|00\rangle$ (control qubit as 0, target qubit as 0)
 "Initial system state: $|00\rangle$ $|00\rangle$: 0.494 $|01\rangle$: 0 $|10\rangle$: 0 $|11\rangle$: 0.506
 Measurements of two qubits matched: 1000, All sum: 1"
- $|01\rangle$ (control qubit as 0, target qubit as 1)
 "Initial system state: $|01\rangle$ $|00\rangle$: 0 $|01\rangle$: 0.536 $|10\rangle$: 0.464
 $|11\rangle$: 0 Measurements of two qubits matched: 0, All sum: 1"
- $|10\rangle$ (control qubit as 1, target qubit as 0)
 "Initial system state: $|10\rangle$ $|00\rangle$: 0.508 $|01\rangle$: 0 $|10\rangle$: 0 $|11\rangle$: 0.492
 Measurements of two qubits matched: 1000, All sum: 1"
- $|11\rangle$ (control qubit as 1, target qubit as 1)
 "Initial system state: $|11\rangle$ $|00\rangle$: 0 $|01\rangle$: 0.496 $|10\rangle$: 0.504
 $|11\rangle$: 0 Measurements of two qubits matched: 0, All sum: 1"

From the above results, there are 2 things that should be seen, probabilities and final value of two qubits.

For probabilities, it is noticeable that sum of all collapsing probabilities (all sum) of $|00\rangle$, $|01\rangle$, $|10\rangle$ and $|11\rangle$ in each case is always 1. Also, the collapsing probability in corresponding case is around 0.5 (50%), like the chance of coin flip; this agrees to principle of entanglement before collapsing.

About the final value of two qubits, this is shown in 'Measurements of two qubits matched'. It is from the final values of 2 qubits $|XX\rangle$ after passing CNOT gate, as shown in Figure 4. There are 2 cases where values are 1000 which is when initial system state is either $|00\rangle$ or $|10\rangle$. It is easy to understand for $|00\rangle$ case since both initial and final qubits are unchanged, so 'Measurements of two qubits' is always true and be counted. In another case that initial qubits are $|10\rangle$; as the control qubit is 1, so the target qubit will be toggled after passing CNOT gate. With this reason,

the final qubits will be $|11\rangle$ which makes 'Measurements of two qubits matched' be true and counted.

V. CONCLUSION

This study demonstrated that quantum entanglement could be created by applying H (Hadamard) and CNOT gates. Calculation using basic matrix was provided to make the topic obvious and simple to understand. To exhibit how to utilize quantum entanglement concepts in IT programming, the Q# language was used to create an entanglement library (.dll). This library was then run on quantum simulator and communicated with the outside world via .NET Web API written in C#. With the help of Web API, various platforms of client such as windows, web, mobile or even IoT apps can have channel, through simulator, to perform quantum works. In this case, by sending control and target qubits to process quantum entanglement computation via Swagger web client, the results from the system were matched with the result from matrix calculation. Q# codes in this study can also be run on a real quantum computer. This article benefits anyone who is interested in learning the basic concepts of quantum entanglement and wants to practice quantum programming via simulator in his or her own computer.

REFERENCES

- [1] B. Marr, "Quantum Computing Now And In The Future: Explanation, Applications, And Problems," *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," *CNBC*, Jun. 06, 2022. <https://www.cnb.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," *Independent*, Oct. 31, 2021. <https://www.independent.co.uk/tech/china-scientists-programmable-quantum-computers-b1946018.html> (accessed Apr. 26, 2023).
- [4] K. Chantarasathaporn, C. Thaiupathump, C. Kama, N. Nopakun, and S. Ngammongkolwong, "Practical Entanglement for Quantum Computing on Quantum Simulator by Q#," in *Proceeding of The 19th International Conference in Applied Computer Technology and Information System*, S. Rungruang and W. Chupradist, Eds., Bangkok: Southeast Bangkok University, Mar. 2023, pp. 408–418.
- [5] K. Kris, "Windows on Quantum Computer (Explained)," *Workwut*, Aug. 22, 2022. <https://workwut.com/windows-quantum-computer/> (accessed Mar. 15, 2023).
- [6] J. Dargan, "Top 5 Quantum Programming Languages in 2022," *The Quantum Insider*, 2022. <https://thequantuminsider.com/2022/07/28/state-of-quantum-computing-programming-languages-in-2022/> (accessed Mar. 15, 2023).
- [7] Z. Hussain and A. Talib, "Strengths and Weaknesses of Quantum Computing," *Int J Sci Eng Res*, vol. 7, no. 9, Sep. 2016.
- [8] A. Tatourian, "Lecture Notes of Quantum Computing for Computer Scientists," *wordpress.com*, Sep. 01, 2018. <https://tatourian.blog/2018/09/01/quantum-computing-for-computer-scientists/> (accessed May 01, 2023).

- [9] H. Paudel, M. Syamlal, and S. Crawford, "Quantum Computing and Simulations for Energy Applications: Review and Perspective," *An Open Access Journal of the American Chemical Society*, vol. 2, no. 3, pp. 151–196, Jan. 2022, Accessed: Mar. 17, 2023. [Online] . Available: <https://pubs.acs.org/doi/10.1021/acseengineeringau.1c00033>