NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

QUANTUM COMPUTING
ON A PHYSICAL QUANTUM COMPUTER

by

Jonathan Kasel

June 2018

Thesis Advisor: Theodore D. Huffmire
Second Reader: James H. Luscombe

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			*Form Approved OMB No. 0704-0188*

1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** June 2018	**3. REPORT TYPE AND DATES COVERED** Master's thesis

4. TITLE AND SUBTITLE
QUANTUM COMPUTING ON A PHYSICAL QUANTUM COMPUTER

5. FUNDING NUMBERS

6. AUTHOR(S) Jonathan Kasel

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
N/A

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release. Distribution is unlimited.

12b. DISTRIBUTION CODE
A

13. ABSTRACT (maximum 200 words)

Several emerging companies are making claims about their quantum computing capabilities. The Department of Defense needs these claims evaluated against known models to objectively compare available systems. This thesis examines the results of tests of quantum circuits on the *ibmqx2* and compares them to our own calculations. This thesis provides a bridge between the quantum circuit model as presented in *Quantum Computing for Computer Scientists* and the compiled circuits provided by IBM. The study is limited to algorithms that can be run with a maximum of five quantum bits.

14. SUBJECT TERMS
quantum computing, IBM, Qubit, cloud computing, quantum physics, superposition, entanglement

15. NUMBER OF PAGES 123

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	**18. SECURITY CLASSIFICATION OF THIS PAGE**	**19. SECURITY CLASSIFICATION OF ABSTRACT**	**20. LIMITATION OF ABSTRACT**
Unclassified	Unclassified	Unclassified	UU

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release. Distribution is unlimited.

QUANTUM COMPUTING ON A PHYSICAL QUANTUM COMPUTER

Jonathan Kasel
Lieutenant, United States Navy
BS, California State University, Sacramento, 2009

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
June 2018

Approved by: Theodore D. Huffmire
Advisor

James H. Luscombe
Second Reader

Peter J. Denning
Chair, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Several emerging companies are making claims about their quantum computing capabilities. The Department of Defense needs these claims evaluated against known models to objectively compare available systems. This thesis examines the results of tests of quantum circuits on the *ibmqx2* and compares them to our own calculations. This thesis provides a bridge between the quantum circuit model as presented in *Quantum Computing for Computer Scientists* and the compiled circuits provided by IBM. The study is limited to algorithms that can be run with a maximum of five quantum bits.

THIS PAGE INTENTIONALLY LEFT BLANK

Table of Contents

THIS PAGE INTENTIONALLY LEFT BLANK

List of Figures

Acknowledgments

I would like to thank first and foremost my wife, Jeanna. Her support and understanding made the process of writing this thesis possible. Her patient willingness to listen to new concepts I grasped and be a sounding board are cherished.

I also would like to thank my advisor, Dr. Ted Huffmire, whose instruction in the field of Quantum Computing was instrumental in writing this thesis. Thank you for the hours you have dedicated to teaching me the fundamentals and making this product the best it could be.

My gratitude also extends to Dr. James Luscombe, my second reader, whose experience helped to elevate and ensure the technical accuracy of the thesis.

I also want to regonize my cohort for patiently listening as I explained what was promised to be "a really interesting aspect of quantum computing."

I am extremely grateful for the IBM team responsible for developing and maintaining the *IBM Quantum Experience*.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 1:
Introduction

On Mar 6, 2017, IBM announced the release of the *IBM Quantum Experience* [2]. The system was a five-qubit quantum computer that the public could interact with through a cloud-based interface. Although five qubits is insufficient to solve real-world problems, it creates an opportunity for experimentation. With the *IBM Quantum Experience*, users are able to run experiments on the current version of the IBM quantum chip (the latest version as of the time of this publication being the *ibmqx5* chip) or a quantum simulator. The quantum simulator is a user-defined configuration capable of simulating up to twenty qubits. The IBM quantum computer and simulator both have a graphical user interface, *Composer*, as well as a coding language, Quantum Assembly Language (*QASM*)[1], which was developed for describing quantum circuits.

The concept of quantum computing was first introduced by Richard Feynman in 1982 [3]. Since then, various companies have entered the market with their version of a quantum computer. There is yet to be a standard defined for a quantum computer or even what tests should be used to benchmark their performance. Despite the ongoing debates on how the physical construction of a quantum computer should proceed, scientists have been working on developing quantum algorithms for those devices. Some of the most well-known algorithms include Deutsch-Jozsa's algorithm, Grover's search algorithm, and Shor's factoring algorithm; we explore them using IBM's quantum experience [4]–[6].

Current quantum computing technology is not attainable for individual consumers. One of the largest barriers is cost. The price of the latest D-wave quantum computer is around $15,000,000 [7]. The current size of most of these computers would also make them difficult for consumer use. Figure 1.1 shows the size of the *ibmqx2*. The *ibmqx2* processor chip, which is seen suspended from the center of the support structure, is comparable in size to the tower of a classical desktop computer. However, the shielding and structure would take up the space of a household bathroom.

Quantum computers that can implement the quantum circuit model are approaching the

[1] QASM is pronounced *kasm*.

theoretical limit of what classical computers can simulate, which is around 56 qubits [8]. Even with a few qubits, as in the *ibmqx2*, it is possible to run simple algorithms demonstrating a quantum speed-up, but these are often limited to example cases that have little practical utility. Boolean operations performed on a classical computer can also be implemented on a quantum computer, but quantum speed-up does not come from the hardware's ability to operate at a faster speed than classical computers. The research team from IBM described the types of problems at which a quantum computer would excel: "quantum computing targets problems that can exploit entanglement to explore correlations in computations, then selects the correct answer through constructive interference" [9]. The speed advantage of a quantum computer is based on the exploitation of quantum mechanics. Purchasing a quantum computer to replace a classical computer would be a very expensive investment with little gain at this time. Finally, there is the issue of calibration. Quantum systems are susceptible to many sources of interference, which makes maintaining coherence difficult. The *ibmqx2* is calibrated twice a day to ensure accuracy; this level of maintenance would be a challenge for a regular computer user to achieve [10].

Figure 1.1. IBM Quantum Computer (*ibmqx2*). Source: [11].

Although the field of quantum computing is not yet at a state where individuals can experiment with their own personal quantum computer, endeavors like the *IBM Quantum Experience* provide accessibility to anyone with an Internet connection. This is, however, an exciting time for large organizations to begin experimenting and investing in quantum information science. Moore's law, which describes the rate of increase in the number of transistors that can fit onto an integrated circuit, has held true for decades. Some speculate that this law has a limited time for being a reliable indicator of advancement in processors [12]. There are many alternatives to increasing the capability of CPUs; parallel computing is one well-documented approach. Quantum computing provides an alternative

type of parallelism. The difficulty of factoring large integers is a cornerstone of asymmetric encryption. Shor's algorithm makes it possible to factor large integers in polynomial time [13]. If Shor's algorithm were implemented on a large enough fault-tolerant quantum computer, it could make some asymmetric ciphers no longer a viable option for protecting confidential communications.

The Department of Defense should continue to invest in quantum computing. The DoD is entering a new age where its military advantages are being contested by other nations; it is vital that the U.S. maintains its technological advantage. The Chinese government has ten billion dollars to construct new research facilities for quantum information science [14]. Quantum computing has the ability to be another "Sputnik moment" for the United States. The DoD must continue engaging with academia and industry to develop scalable quantum computers. Whether it is to stay in front of cryptographic challenges or to pursue solutions to computational problems previously unrealistic to compute, the DoD needs to maintain a notable presence in the field of quantum computing.

1.1 What is the *IBM Quantum Experience*?

The *IBM Quantum Experience* was designed as an open and interactive platform, providing tools for experimentation and process refinement [10]. After signing up for an account, a user is given a set of currency units which allow for experimentation. When experiments are run on one of the available quantum processors, the user's units are reduced by an amount dependent on the complexity of the experiment. Once the experiment has gone through the queue and been performed, the units are again available to the user to run additional experiments.

The *ibmqx2* went online on January 24, 2017 [10]. The quantum chip is constructed of fixed-frequency superconducting transmon.[2] The chip is stored inside a dilution refrigerator that keeps the processor at approximately 15 milli-Kelvin. The chip is shielded from external variables such as light, heat, and magnetic interference. Not all qubits are fully connected, meaning that when a user applies a Controlled-NOT (***CNOT***) to qubit a as the control qubit and to qubit b as the target qubit, the choices for qubit b are limited by the physical

[2]A transmon is "a transmission-line shunted plasma oscillation qubit" [15]. This form of qubit aims to increase decoherence times by operating at an optimal working point [15].

implementation of the specific processor. The connectivity of the *ibmqx2*, achieved by two coplanar waveguides, is depicted in Figure 1.2.

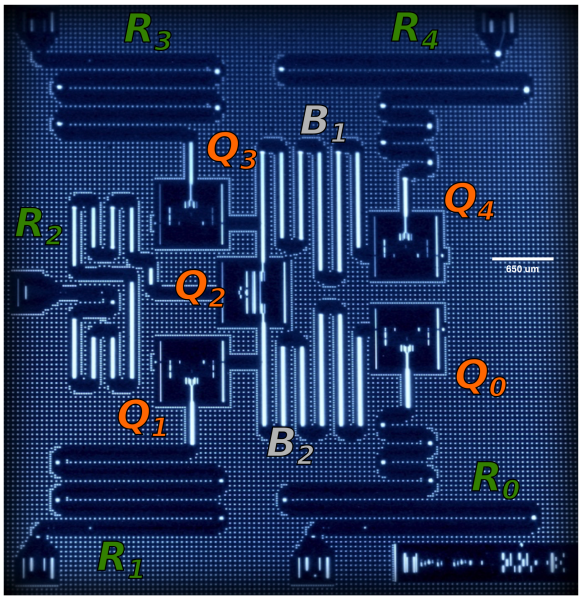

Figure 1.2. *ibmqx2* processor. Source: [16].

The arrows in Figure 1.3 point from the control qubit to the available targets. For example, q_0 is able to target both q_1 and q_2, but is not able to be the target of any qubits.

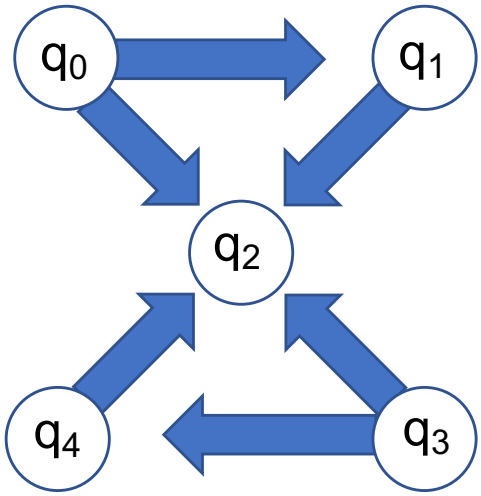

Figure 1.3. *ibmqx2* processor connectivity. Adapted from [16].

The specific configuration of control and target qubits is not consistent between all versions of IBM's quantum processors. The quantum scores generated are not directly backward-compatible. It is possible, as shown in Chapter 3, to use swap gates to adapt a circuit developed for one version of the processor to another. We recommend the development of a higher-level programming language to map between specific versions of the quantum processors. This concept is further developed in Chapter 7.

1.2 Introduction to Notation

The *IBM Quantum Experience* uses multiple different forms of notation to represent the quantum algorithms, states, and results. It is helpful to first represent a quantum bit in an arbitrary state of superposition. From this foundation it is possible to develop more specific examples.

$$|\Psi\rangle = \alpha|0\rangle + \beta|1\rangle \tag{1.1}$$

The form above is known as Dirac notation or bra-ket notation. The latter refers to the recommendation by Paul Dirac that the '$\langle$' symbol be called a *bra* and the '$\rangle$' called a *ket* [17]. The coefficients α and β are complex numbers know as probability amplitudes for the qubit taking on the discrete state of $|0\rangle$ or $|1\rangle$, respectively. They are defined by Equation 1.2.

$$|\alpha^2| + |\beta^2| = 1 \tag{1.2}$$

Equation 1.2 shows that the sum of the probabilities must equal 1. With this background it is helpful to examine a discrete ground state of a qubit. A qubit in a $|0\rangle$ state would be represented in Dirac notation as shown Equation 1.3.

$$|0\rangle = 1|0\rangle + 0|1\rangle \tag{1.3}$$

This means the qubit will be in a $|0\rangle$ state with a probability of 1 when measured.

Another common notation to represent a qubit is through the use of a column vector. This is particularly helpful when examining the effect a particular transformation will have as represented by a unit circle or Bloch sphere; see Chapter 2. To represent the arbitrary state from Equation 1.1 as a column vector, simply take α to be the first element and β to be the second: $\begin{bmatrix}\alpha\\ \beta\end{bmatrix}$. The same is done to represent $|0\rangle$ as a column vector, with a result of $\begin{bmatrix}0\\ 1\end{bmatrix}$.

1.3 Matrix Multiplication

This thesis evaluates several well-known quantum algorithms on the specific IBM architecture. One way to evaluate two different algorithms is to compile the various transformations into a single unitary matrix. This process involves matrix multiplication. There are many software packages readily available to perform this calculation, but a basic understanding is helpful for understanding the concept and with troubleshooting.

We are given matrices A and B, where A is a 2-by-3 matrix, and B is a 3-by-2 matrix as defined in Equation 1.4.

$$A = \begin{bmatrix} a & b & c \\ d & e & f \end{bmatrix}, B = \begin{bmatrix} g & h \\ i & j \\ k & l \end{bmatrix} \tag{1.4}$$

The product of the two matrices is $C = AB$, where $C_{ij} = \sum_h A_{ih} B_{hj}$.

$$AB = \begin{bmatrix} (ag + bi + ck) & (ah + bj + cl) \\ (dg + ei + fk) & (dh + ej + fl) \end{bmatrix} \tag{1.5}$$

It is important to note that the number of columns of matrix A must be equal to the number of rows of matrix B. Matrix multiplication is not a commutative operation, i.e., $AB \neq BA$. This can be seen in Equation 1.5, where matrix AB is a 2-by-2 matrix, and BA is a 3-by-3 matrix; therefore AA would be undefined in this example. Scalar multiplication is similar to the distributive property of multiplication. In this case it is helpful to look at our first single-qubit quantum gate, the Hadamard gate, or $\boldsymbol{H}$ gate. The Hadamard matrix is defined in Equation 1.6 [1], [18].

$$\boldsymbol{H} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \tag{1.6}$$

Using scalar multiplication, the $\frac{1}{\sqrt{2}}$ can be distributed to each coefficient in the matrix resulting in Equation 1.7.

$$\boldsymbol{H} = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{-1}{\sqrt{2}} \end{bmatrix} \tag{1.7}$$

1.3.1 Tensor Products

A tensor product can be used to construct a vector of higher dimensions, for example, the two bit state $|10\rangle$. This notation is most frequently written in base two. Throughout this thesis, only base two will be used in conjunction with Dirac notation. The tensor product is frequently denoted by the $\otimes$ symbol. Equation 1.8 shows the column vector for $|10\rangle$. This column vector will also conform to Equation 1.2.

$$|1\rangle \otimes |0\rangle = |1\rangle|0\rangle = |10\rangle = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \tag{1.8}$$

In Equation 1.8, there were two qubits in the system: the first, q_0, in a $|1\rangle$ state and the second, q_1, in a $|0\rangle$ state. Equation 1.9 changes the number of qubits in the system to three. This shifts the qubit from Equation 1.8, which is in a $|1\rangle$ state, from q_0 to q_1 and adds a third qubit, q_2, which is in a $|0\rangle$ state.

$$|010\rangle = |0\rangle \otimes |1\rangle \otimes |0\rangle = |0\rangle|1\rangle|0\rangle = |010\rangle = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{1.9}$$

1.3.2 Kronecker Product

With a basic understanding of tensor products, matrix multiplication, and scalar multiplication, it is possible to understand the Kronecker product. This is simply a more specific example of the tensor product discussed in Section 1.3.1. The notation used in a Kronecker product is the same as a tensor product. The Kronecker product, also known as the outer product, of two matrices A and B is written as $A \otimes B$. Equation 1.11 demonstrates how to take the Kronecker product. One way to think of this operation is apply each coefficient from matrix A to matrix B through scalar multiplication.

$$A = \begin{bmatrix} a_{0,0} & a_{0,1} \\ a_{1,0} & a_{1,1} \end{bmatrix} \quad B = \begin{bmatrix} b_{0,0} & b_{0,1} \\ b_{1,0} & b_{1,1} \end{bmatrix} \tag{1.10}$$

$$
\begin{aligned}
A \otimes B &= \begin{bmatrix} a_{0,0} \begin{bmatrix} b_{0,0} & b_{0,1} \\ b_{1,0} & b_{1,1} \end{bmatrix} & a_{0,1} \begin{bmatrix} b_{0,0} & b_{0,1} \\ b_{1,0} & b_{1,1} \end{bmatrix} \\ a_{1,0} \begin{bmatrix} b_{0,0} & b_{0,1} \\ b_{1,0} & b_{1,1} \end{bmatrix} & a_{1,1} \begin{bmatrix} b_{0,0} & b_{0,1} \\ b_{1,0} & b_{1,1} \end{bmatrix} \end{bmatrix} \\
&= \begin{bmatrix} (a_{0,0} \times b_{0,0}) & (a_{0,0} \times b_{0,1}) & (a_{0,1} \times b_{0,0}) & (a_{0,1} \times b_{0,1}) \\ (a_{0,1} \times b_{1,0}) & (a_{0,1} \times b_{1,1}) & (a_{0,1} \times b_{1,0}) & (a_{0,1} \times b_{1,1}) \\ (a_{1,0} \times b_{0,0}) & (a_{1,0} \times b_{0,1}) & (a_{1,1} \times b_{0,0}) & (a_{1,1} \times b_{0,1}) \\ (a_{1,0} \times b_{1,0}) & (a_{1,0} \times b_{1,1}) & (a_{1,1} \times b_{1,0}) & (a_{1,1} \times b_{1,1}) \end{bmatrix}
\end{aligned}
\tag{1.11}
$$

By taking the Kronecker product and using matrix multiplication it is possible to represent a quantum circuit as a single matrix. This concept is one of the primary techniques used in this thesis to compare the examples of well-known algorithms provided in the *IBM User Guide* to those as presented in *Quantum Computing for Computer Scientists* [18]. The results of this examination should provide an additional indicator to confirm that the *ibmqx2* follows the quantum circuit model.

1.4 Quantum Laws

The *IBM User Guide* describes many of the concepts related to quantum computing [10]. It also provides example experiments to demonstrate many of these concepts on their physical hardware. IBM lists five key laws that help to define quantum physics:

- Quantum is a system like everything else.
- A quantum state is a configuration of the system.
- A quantum state changes; it naturally wants to evolve, (i.e., quantum gates must be reversible)
- Scaling - how parts make a whole.
- Quantum measurements are probabilistic. [10]

CHAPTER 2:
Quantum Computing Basics

2.1 Quantum Gates

Chapter 1 provided the necessary background to begin constructing mathematical representations of quantum gates as a matrix. Quantum gates must have the property of being reversible [18]. Gates such as the ***AND*** gate lose information, i.e., by Landauer's principle, they dissipate energy (see Equation 2.2). This principle unites information to the second law of thermodynamics and is an established concept in quantum computing [19]. For additional reading on the subject, see Chapter 5 of [18]. The important concept for a computer scientist to understand is that quantum gates must be logically reversible: "A device is said to be logically irreversible if its input cannot be uniquely determined from its output" [20].

$$\boldsymbol{AND} = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.1}$$

$$\boldsymbol{AND}|10\rangle = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \tag{2.2}$$

It is clear from Equation 2.2 that information was lost. The previous state of the system could have been $|00\rangle$, $|01\rangle$, or $|10\rangle$. There is no way to determine what state the two-qubit system was in prior to the ***AND*** gate being applied.

We have already seen one example of a single-qubit quantum gate, the Hadamard gate, in Section 1.3. The Hadamard gate will be further discussed in Section 2.3.

2.1.1 Identity Gate

The next single-qubit gate is the Identity gate, or $\boldsymbol{I}$ gate. The gate, as expected, returns the original state of the system prior to the gate being applied.

$$\boldsymbol{I} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{2.3}$$

$$\boldsymbol{I}|0\rangle = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \tag{2.4}$$

$$\boldsymbol{I}|1\rangle = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \tag{2.5}$$

The $\boldsymbol{I}$ gate is useful when compiling a quantum circuit. It is common in quantum algorithms to only apply a quantum gate to a few qubits at a time. It is not necessary to show the $\boldsymbol{I}$ gate, as it is understood that applying the $\boldsymbol{I}$ gate is functionally equivalent to not applying any gate.

2.1.2 Pauli Gates

The Pauli matrices are named after Wolfgang Pauli and are commonly used in quantum computing [1]. Each of the three gates below performs a rotation around a different spin axis. The three gates are the $\boldsymbol{X}$, $\boldsymbol{Y}$, and $\boldsymbol{Z}$ gates. The $\boldsymbol{I}$ gate is occasionally included with these gates due to the shared mathematical properties, and may be referenced as σ_0. The $\boldsymbol{X}$ gate performs a bit flip on a qubit, and is equivalent to a classical ***NOT*** gate. Each of the Pauli matrices performs a similar flip, or 180 degree rotation, of the qubit about its respective axis.

$$\boldsymbol{X} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \tag{2.6}$$

$$\boldsymbol{Y} = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} \tag{2.7}$$

$$\boldsymbol{Z} = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \tag{2.8}$$

2.1.3 Clifford and Non-Clifford Gates

The Pauli matrices listed above are part of a special group of gates known as the Clifford group. This group is covered under the Gottesman-Knill theorem of gates that can be efficiently simulated with a classical computer [21]. The theorem also includes the Hadamard gate, Controlled-NOT (***CNOT***) gate, and a phase shift gate as a part of the Clifford Group. The Hadamard gate will be covered in Section 2.3 and the ***CNOT*** in Section 2.2. The phase shift gate is also called the ***S*** gate.

$$\boldsymbol{S} = \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix} \tag{2.9}$$

The Non-Clifford Gates are needed to make the *ibmqx2* universal [10], [22]. IBM implemented the ***T*** gate for this purpose.

$$\boldsymbol{T} = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{bmatrix} \tag{2.10}$$

2.2 Multiple Qubit Gates

Controlled-NOT

Up to this point, only single-qubit gates have been introduced. In order to construct interesting algorithms, it is necessary to implement conditional logic. The ***CNOT*** gate enables this. The basic ***CNOT*** operates on two qubits, q_0 and q_1. The ***CNOT*** gate applies an ***X*** gate to the target qubit, q_1, if the control qubit, q_0, is $|1\rangle$. In Figure 2.1, the target qubit can be identified by the small solid circle, and the target qubit may be identified by the $\oplus$ symbol. When reading a quantum circuit in this form, pay close attention to the location of these symbols. A reverse ***CNOT***, shown in Figure 2.2, is not the equivalent to a ***CNOT***, shown in Figure 2.1.

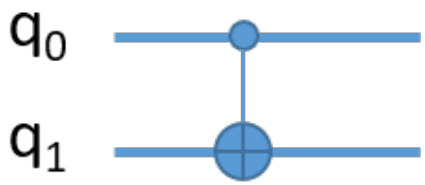

Figure 2.1. ***CNOT*** gate.

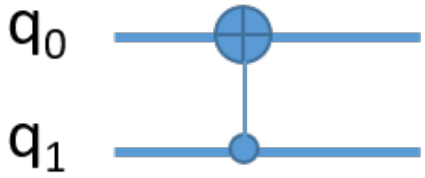

Figure 2.2. Reverse ***CNOT*** gate.

The matrix for the ***CNOT*** gate from Figure 2.1 is shown below. Because the qubits are directly adjacent to each other, the matrix defining this gate is a 2-by-2 matrix. This is not the case if the target bit is separated from the control by another bit.

$$\boldsymbol{CNOT} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{2.11}$$

An example ***CNOT*** circuit with a control bit q_0 and a target bit q_2 is shown in Figure 2.3. This implements the same logic as that in Figure 2.1, but has changed the target qubit.

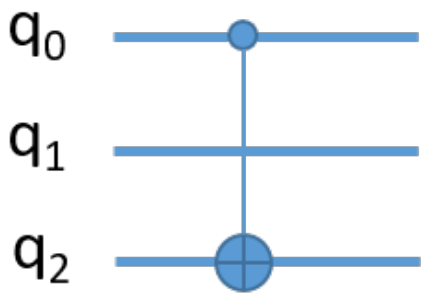

Figure 2.3. ***CNOT*** gate with three qubits.

The resulting matrix must account for q_1, which is not changed in applying this operation. Yanofsky explains the general procedure for determining matrices such as the one in Equation 2.12.

$$\boldsymbol{CNOT}_{02} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \tag{2.12}$$

Toffoli Gate

The gates defined thus far make the *ibmqx2* universal but not yet functionally complete [10]. The Toffoli gate provides the last component for functional completeness. The Toffoli gate can be used to construct reversible ***AND***, ***NAND***, and ***OR*** gates [1], [18]. The logic of a Toffoli gate is similar to that of a ***CNOT***, but instead of one control qubit, there are two. Figure 2.4 shows a Toffoli gate with q_0 and q_1 as the two control qubits and q_2 as the target qubit. Equation 2.13 shows the matrix that represent the Toffoli gate.

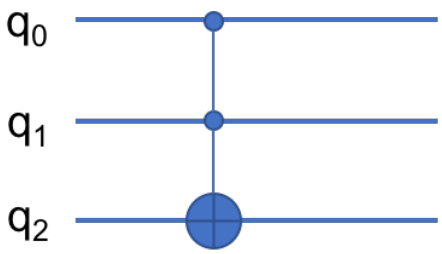

Figure 2.4. Toffoli gate.

$$Toffoli = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \tag{2.13}$$

IBM does not provide a compiled Toffoli gate as an option for use on the *ibmqx2*. The gate can be implemented by decomposing the single-qubit gates and ***CNOT*** gates as shown in the *IBM User's Guide* [10] or through the use of *QASM* code, shown as an example in [23].

2.3 Superposition

To begin to understand the concept of superposition, it is helpful to review Thomas Young's double-slit experiment [18]. His experiment has been recreated many times since he presented his results in 1803. Young's experiment demonstrated the wave nature of light. The experiment has two walls: one is a sensor, and the other has two slits in it. A wave is then sent towards the wall with slits; as it passes through, it moves towards the sensor. The wave refracts and then constructively and destructively interferes with itself. This creates a unique pattern on the sensor that is predictable given the features of the wave.

Richard Feynman introduces the double-slit experiment in two primary ways as a basis for comparison with conducting the experiment with electrons, one using waves and another

example with bullets [24]. He first describes the experiment as using a large particle, bullets. The distribution of where the bullet strikes the backstop is shown as a smooth curve after the process is repeated multiple times; Figure 2.7 depicts the experiment and results. There is no evidence of the bullet interfering with itself as it passes through the slit. Figure 2.6 illustrates the double-slit experiment with a wave. This experiment clearly shows the impact the interference has on the probability of the wave's height when it is detected.

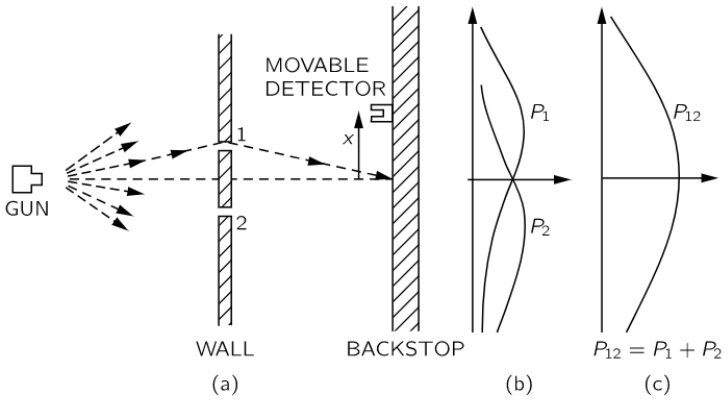

Figure 2.5. Example of Young's double-slit experiment with bullets. Source: [24].

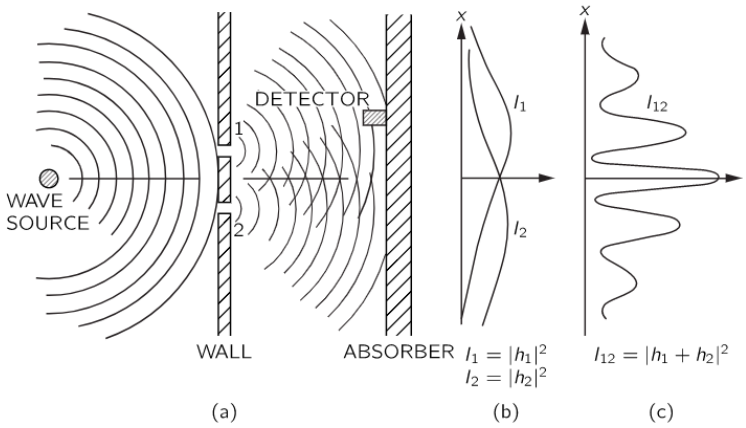

Figure 2.6. Example of Young's double-slit experiment with waves. Source: [24].

Light was first accurately described by Einstein in 1905 [25]. Taking the same initial setup,

but using individual photons shot through the slits, the same interference pattern as seen with a waves is observed with single photos. Feynman describes conducting this experiment in theoretical terms and advised that you "should not try to set up this experiment...The trouble is that the apparatus would have to be made on an impossibly small scale" [24]. Figure 2.7 illustrates his description of the double-slit experiment using electrons. Since Feynman's lecture series, technological advances have encouraged several variations of this experiment to be performed in a laboratory environment [26], [27]. These results experimentally demonstrate superposition in naturally occurring systems. The only way to explain the distribution is that each single photon is passing through both slits simultaneously and interfering with itself: the photon is in a state of superposition.

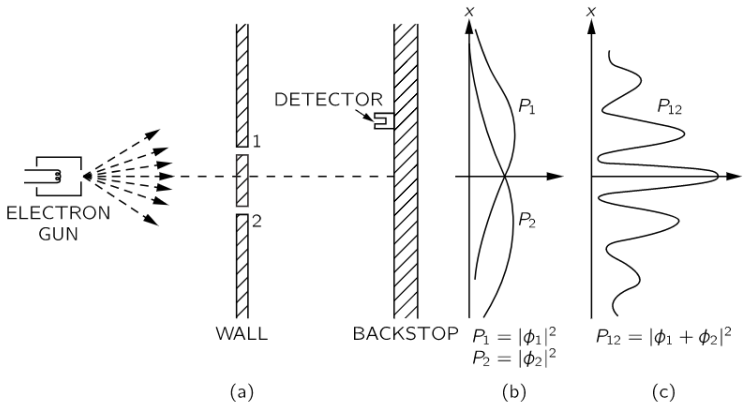

Figure 2.7. Example of Young's double-slit experiment with electrons. Source: [24].

The concept of superposition is one of the fundamental properties of quantum computing. Yanofsky describes superposition as a "haziness" of the physical properties. "Rather than an object's being in one position or another, we say that it is in a 'superposition,' i.e., in some sense, it is simultaneously in more than one location at the same time [18]." This superposition also applies to the spin of a quantum bit. The spin is measured along a certain axis and then interpreted to be in a state of $|0\rangle$ or $|1\rangle$.

This concept may not immediately seem all that powerful, but when you begin to examine larger systems, it becomes clear that a superposition of a system with N qubits would have 2^N possible states. The $\boldsymbol{H}$ gate from Chapter 1 and the $\boldsymbol{S}$ gate from Section 2.1.3 are both

able to place a qubit into superposition. It is simple to see this in Equation 2.14.

$$\boldsymbol{H}|0\rangle = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{-1}{\sqrt{2}} \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix} \tag{2.14}$$

If we apply Equation 1.2 to Equation 2.14, it is clear that there is a 50% probability that this qubit will be in a $|0\rangle$ state and a 50% probability that this qubit will be in a $|1\rangle$ state after measurement. Observation of the quantum state collapses the superposition, causing each qubit to behave classically. Each qubit will be in a definite state of $|0\rangle$ or $|1\rangle$.

2.4 Entanglement

The topic of entanglement has an interesting history in the field of physics. Einstein wrestled with this concept, and his discourse on the subject spawned his famous description of entanglement as "spooky action at a distance" [28]. IBM describes this concept as follows: "two systems that appear too far apart to influence each other can nevertheless behave in ways that, though individually random, are too strongly correlated to be described by any classical local theory" [10]. Entangled states are often referred to as an *EPR* pair, referencing Einstein, Podolsky, and Rosen's paper released in 1935 which describes what they viewed as a paradox created by entanglement [29]. The paper proposed that there were hidden variables which would account for the correlation between the two states. Quantum mechanics refutes this, as stated by Greenberger, Horne, and Zeilinger:

"If one measures the spin of particle 1, far from the decay point, and finds spin up, say, then one knows with certainty that particle 2, which is far away, has spin down. According to the EPR argument, since one has in no way disturbed particle 2, then this feature, spin down, must be an element of physical reality. Therefore having spin down is a property of the particle itself, and cannot have been produced by any measurement we made on particle 1. It must have come away from the point of interaction, the decay point, with spin down. Quantum mechanics denies this simple point. It says that the spin of particle 2 is indeterminate until the spin of particle 1 is measured" [30].

Equation 2.15 shows the vector which describes an *EPR* or *Bell* state. The term *Bell* state comes from the Bell inequality, originally proposed by John Bell in 1964 [31] and later

refined in [32]. An example Bell state is $|\beta_{00}\rangle$. The state $|\beta_{00}\rangle$ can be created by initializing two qubits in a $|0\rangle$ state. An ***H*** gate is applied to the first qubit, placing it into superposition. A ***CNOT*** is applied to the system as a whole, as shown in Figure 2.8, creating an entangled state.

$$\frac{|00\rangle + |11\rangle}{\sqrt{2}} = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ 0 \\ 0 \\ \frac{1}{\sqrt{2}} \end{bmatrix} = |\beta_{00}\rangle \tag{2.15}$$

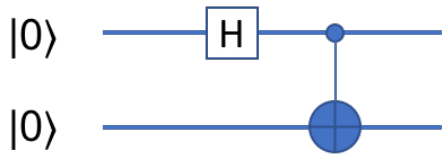

Figure 2.8. Creating the *Bell* state $|\beta_{00}\rangle$.

Bell's theorem demonstrated a violation of an inequality, but there is a stronger proof in the GHZ states [10], [30], [33]. "Instead of a probabilistic violation of an inequality, the GHZ states lead to a deterministic violation of an equality" [10]. An example GHZ state is $\frac{1}{\sqrt{2}}(|000\rangle - |111\rangle)$ [10]. The problem solved by the GHZ state is to find the values satisfying the identities shown below [10].

1. XXY = 1.
2. YXY = 1.
3. YYX = 1.
4. XXX = -1.

The possible values for X and Y are either +1 or -1. After reviewing [30], Mermin recanted his statement that "no set of experiments, real or *gedanken*, was known that could produce such an all-or-nothing demolition of the elements of reality" [33].

2.5 Teleportation

Teleportation in quantum computing "is the process by which the state of an arbitrary qubit is transferred from one location to another" [18]. This might sound like a topic better left for science fiction, but the concept has been demonstrated in laboratory experiments [18]. The key concepts to retain from quantum teleportation are:

- Quantum teleportation can transfer information; it does not make a copy [18].
- Quantum teleportation requires a classical communication channel [1].
- Because of the classical communication channel, teleportation is constrained by the speed of light [1].
- Particles themselves do not move in quantum teleportation, but the state which is created is indistinguishable from the original [18].

At the time of publishing this thesis, the conditional ***IF*** gate had not been implemented on the *ibmqx2*. A simulation of quantum teleportation on the *ibmqx2* is provided in [23].

2.6 Bloch Sphere

Quantum physics frequently uses a Bloch sphere to represent the state of a qubit. The Bloch sphere shown in Figure 2.9 can be used to visualize the state of a qubit.

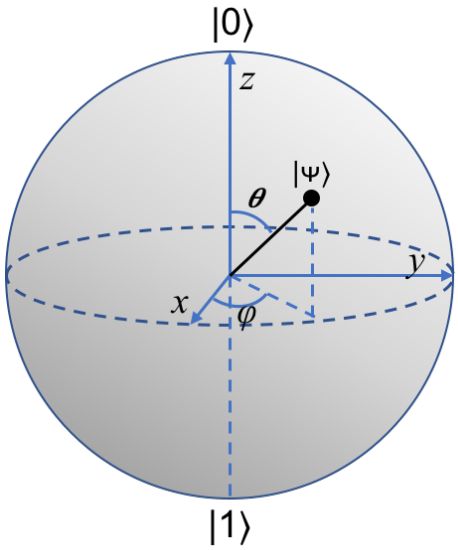

Figure 2.9. Bloch sphere. Adapted from [1].

With the Bloch sphere, it is possible to describe a qubit with only the two parameters θ and ϕ. The x, y, and z values can then be described by Equations 2.16 through 2.18 [18].

$$x = \cos\phi \sin 2\theta \tag{2.16}$$

$$y = \sin 2\theta \sin\phi \tag{2.17}$$

$$z = \cos 2\theta \tag{2.18}$$

CHAPTER 3:
IBM Quantum Experience

3.1 Introduction to *IBM Quantum Experience*

The *ibmqx2* is located in the IBM Headquarters at Thomas J. Watson Research Center, Yorktown NY [11]. Standard users have access to two separate quantum processors, the *ibmqx2* and the *ibmqx4*, as well as two quantum simulators. There are currently two additional quantum computers, the *ibmqx5* (16-qubits) and the *QS1_1* (20-qubits). These are available to researchers through special request[3], and through joining the commercial *IBM Q* network, respectively.

As mentioned in Chapter 1, the *IBM Quantum Experience* has several elements, including a GUI for programming the *ibmqx2*, a text editor to program quantum circuits using *QASM*, a *Beginner's guide*, a *Full User Guide*, a community forum page, instructional videos, and a well-established *GitHub* page of resources.

3.2 Introduction to Composer

Composer is the GUI which enables a user to drag and drop quantum gates to experiment with a quantum computer. IBM refers to the series of gates applied to the qubits as a *quantum score* because of its resemblance to a sheet of music [10]. Figure 3.1 shows a blank *quantum score* after navigating to the *Composer* tab.

[3]Access to the 16-qubit *ibmqx5* was requested while conducting research for this thesis and granted by IBM. Use of the *ibmqx5* is only available through the QISKIT used with Python. The *ibmqx5* was not experimented with in this thesis.

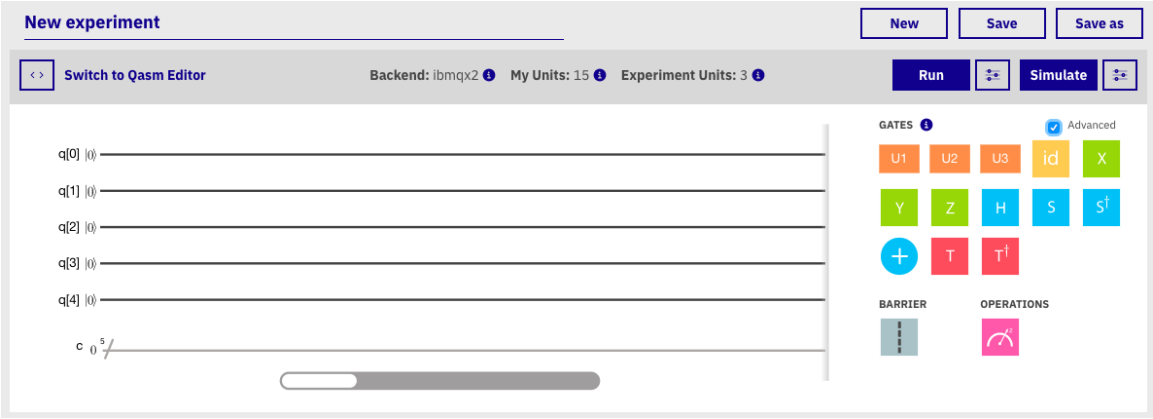

Figure 3.1. *IBM Quantum Experience* composer view. Source: [34].

The *Composer* view shows which processor has been selected; *ibmqx2* in this instance, as noted in the *Backend* field at the top of the score. The next two fields, *My Units* and *Experiment Units*, show the units available to the user and the unit cost for the experiment currently being worked on. In experimenting with the *ibmqx2*, even with the maximum number of gates being applied, the experiment cost remained constant at three units. The maximum number of gates that can be applied in a given quantum score is 85, which includes five measurements at the end of the score for each qubit. The number of units does increase with the number of shots; this is the number of times a given circuit is run. The default parameter for shots is 1024, but it can be increased to 4096 and 8192, as well as lowered to a single shot. Both 4096 and 8192 shot parameters increase the units per experiment to five.

Most of the gates on the right hand side of the composer view were discussed in Chapter 2. The five gates not mentioned are the $\boldsymbol{U}$ gates, the $S^\dagger$ gate, and the $T^\dagger$ gate. The $\dagger$ symbol denotes the transposed conjugate of a quantum gate. Equations 3.1 and 3.2 show the matrices for these gates. The three $\boldsymbol{U}$ gates are the only physically implemented gates and are used to construct the additional gates supported by the *IBM Quantum Experience* [23]. These gates, along with ***CNOT***, form a universal set for defining the rest of the quantum logic gates, such as the $\boldsymbol{X}$, $\boldsymbol{Y}$, and $\boldsymbol{Z}$, in Chapter 2. These gates are only visible if the *Advanced* box is checked. The gates are provided to permit additional user-defined gates.

$$S^{\dagger} = \begin{bmatrix} 1 & 0 \\ 0 & -i \end{bmatrix} \tag{3.1}$$

$$T^{\dagger} = \begin{bmatrix} 1 & 0 \\ 0 & e^{-i\pi/4} \end{bmatrix} \tag{3.2}$$

3.2.1 Building a Quantum Score

To build a quantum score from the *Composer* GUI, click and drag the desired gate to the necessary position on the quantum score. While moving the gate, dots appear on the score highlighting the available locations where the gate can be applied (see Figure 3.3).

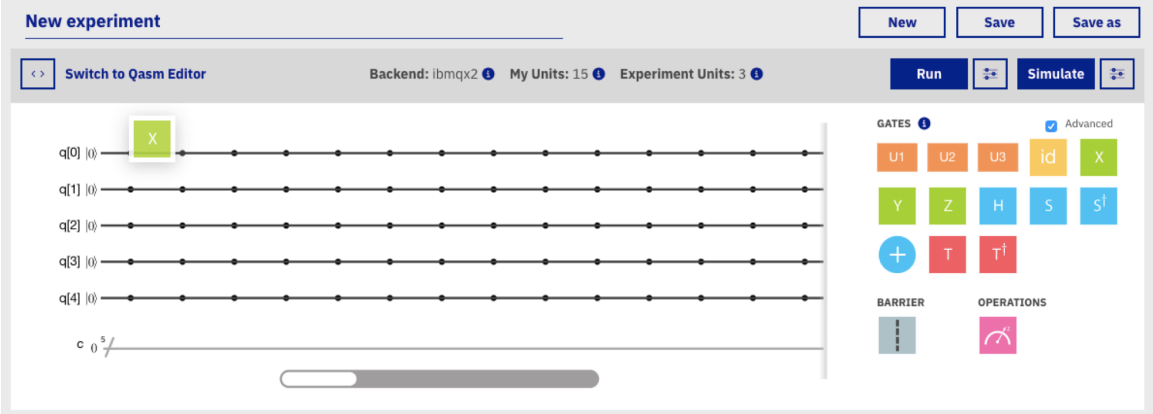

Figure 3.2. Placing an ***X*** gate.

When the mouse is released, the gate will snap to the closest available position. If a gate was incorrectly placed, the process can be repeated by selecting the gate from within the quantum score and dragging it to the correct location. Selecting a gate within the quantum score will adjust the information available on the right hand side, providing specifics about the highlighted gate. To remove a gate, click and drag the gate to the red *delete* box which appears while moving a gate within the quantum score.

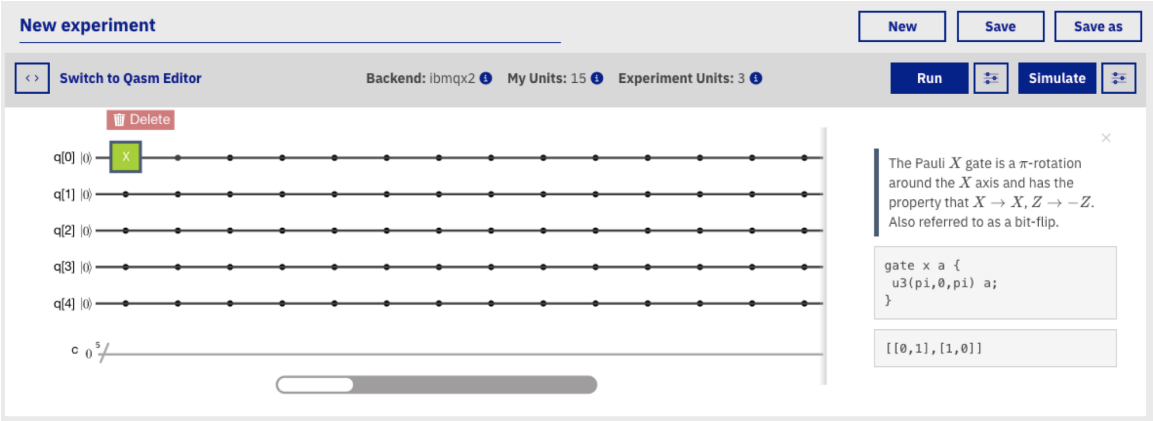

Figure 3.3. Deleting an ***X*** gate.

Adding a ***CNOT*** gate is performed in a similar fashion. The blue $\oplus$ can be selected and dragged to the desired location, but is only able to be applied to qubits q_1, q_2, and q_4. The *Composer* will highlight the score for the qubits which may be designated as targets, as shown in Figure 3.4. The specifics for which qubit may be a target and control is determined by the physical configuration of the qubits, as shown in Figure 1.3[4]. Once the target qubit has been selected by placing the $\oplus$, a small vertical blue bar may be dragged to any of the available control qubits.

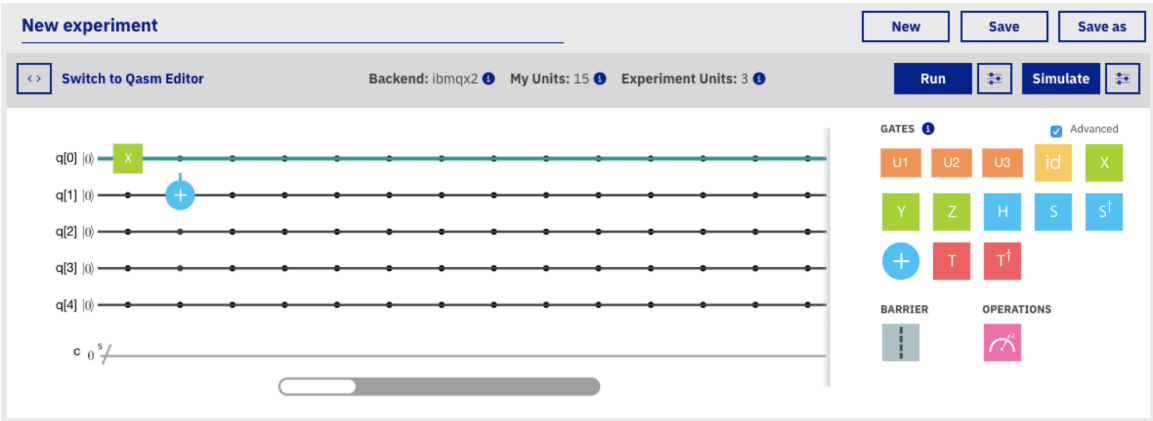

Figure 3.4. Applying a ***CNOT*** gate.

[4]The decision for which qubits were to be targets or controls was made by comparing the relative frequencies of the qubits. "This interaction [when applying a ***CNOT***] is stronger when choosing the qubit with higher frequency to be the control qubit, and the lower frequency qubit to be the target" [16].

3.3 Introduction to QASM

As an alternative to using *Composer* to build quantum circuits, IBM developed *QASM* to allow programming a quantum processor. As the name suggests, *QASM* is similar to a standard low-level programming language. The *IBM Full User Guide* focuses primarily on building quantum circuits through the *Composer* menu. Full documentation on using *QASM* is provided in [23].

The code shown below describes the same circuit shown in Figure 3.5. The *QASM* appears on the left, and the corresponding score on the right. The quantum score is automatically generated while using *QASM* and cannot be manipulated, as described in Section 3.2. The first three lines of the code form the header for the source code. Line one includes the necessary library for the computer to compile the circuit. The next two lines define the number of qubits in the system or the quantum register, `qreg[]`, and the number of classical bits required for the measurement, `creg[]`. Lines five and six instruct the compiler to first apply an ***X*** gate to `q[0]` and then apply a ***CNOT*** gate to `q[0]` as the control and `q[1]` as the target.

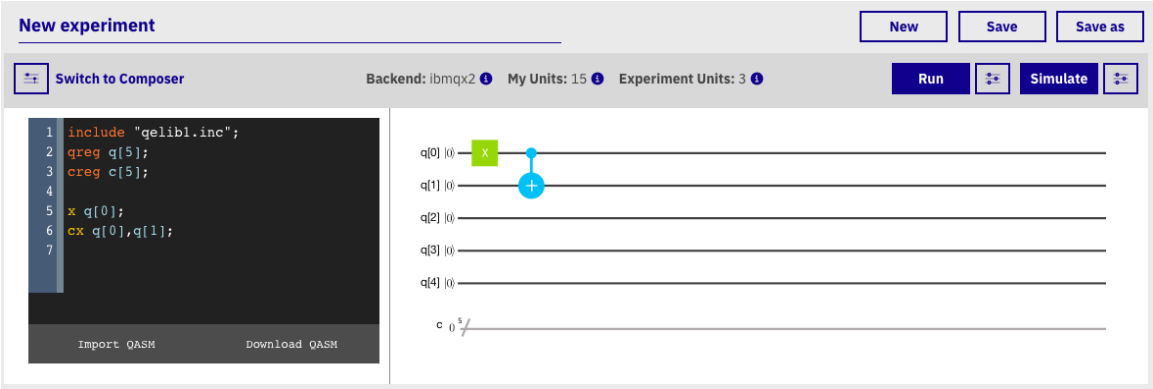

Figure 3.5. QASM example. Source: [34].

This code is not yet ready to run on the *ibmqx2*. Every quantum score requires at least one measurement to run. The measurement can be dragged and dropped like all other gates in the *Composer* or by using *QASM* construct `measure q[] -> c[];`. The default when applying the measurement to a qubit from *Composer* is to map to resulting $|0\rangle$ or $|1\rangle$ state from `q[0]` to `c[0]`, `q[1]` to `q[1]`, and so on. This standard format interprets `q[0]` as the least significant bit and `q[4]` as the most significant bit. The default measurement on the

ibmqx2 is shown in Figure 3.6, and a customized measurement is shown in Figure 3.7.

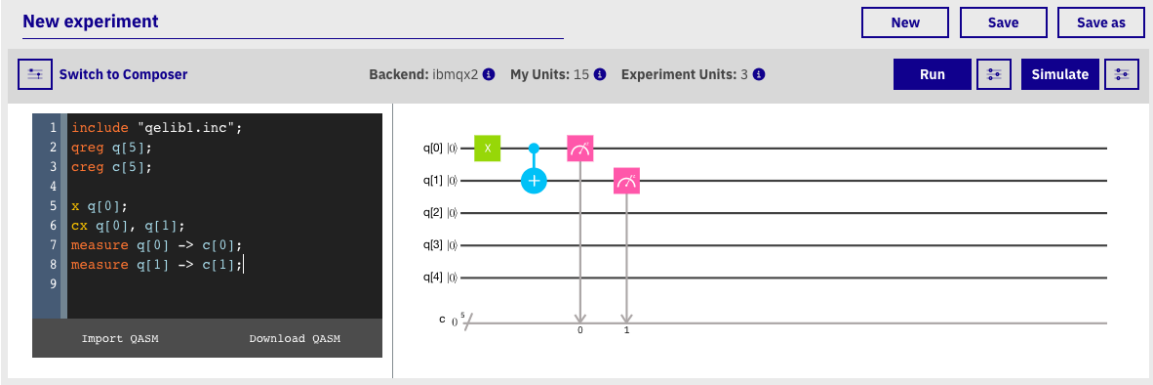

Figure 3.6. Quantum circuit with default measurement. Source: [34].

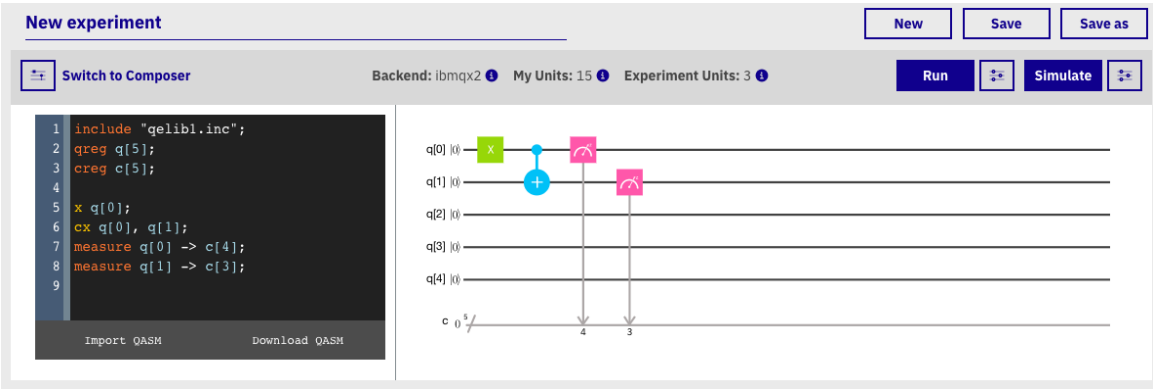

Figure 3.7. Quantum circuit with custom measurement. Source: [34].

The classical bit to which a qubit is mapped at the time of measurement does not impact its state, but the user must be aware of how the results are represented for comparison. Figure 3.6 would result in `00011` in the computational basis, and Figure 3.7 would result in `11000`. The quantum score provides an easy method to verify where each qubit is mapped by the measurement. The small integer below the line adjacent to the classical register corresponds to the classical bit to which the qubit will be mapped.

3.3.1 Quantum Circuit Execution

IBM divides the process by which a quantum circuit is executed into four distinct phases [23]. The phases are:

- Compilation
- Circuit Generation
- Circuit Execution
- Post-Processing

A diagram of the processing a quantum circuit undergoes is provided in Figure 3.8. The "dashed vertical lines separate offline, online, and real-time processes" [23]. The user requesting a circuit to be run is in the left-hand side of Figure 3.8; the center portion is completed by the IBM computer to optimize the circuit and provide necessary run-time parameters. The code is then placed into the appropriate queue by the resource manager until execution which takes place in the right-hand side of the figure.

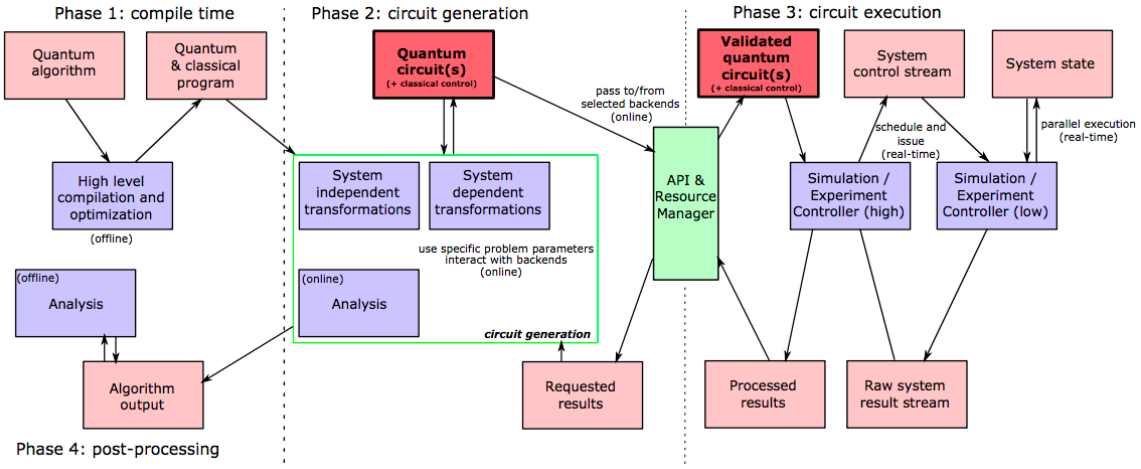

Figure 3.8. Quantum circuit processing and execution. Source: [23].

3.4 Demonstrating Quantum Principles

Before exploring the ability of the *ibmqx2* to demonstrate quantum principles, we will begin with some classical operations to use as benchmarks. The experiments are compared to the *ideal simulator*, which computes the expected result of the score. As stated in Chapter 1, the *ibmqx2* is calibrated twice daily; the most recent calibration results are provided with each result for download. The necessity for these frequent calibrations will be evident in the first two experiments with the *ibmqx2*. The experiments performed all use a standard number of shots for both the *ibmqx2* and the *ideal simulator*, 1024 and 1000, respectively.

Any deviation from this will be specifically mentioned.

Experiment 3.1: Measuring the Initial State

The first quantum score will measure the qubits with no gates applied. This gives a starting point to understand the error levels of each qubit. The quantum score in Figure 3.9 shows the measurement on each qubit with no additional gates. The expectation is that because each qubit is initially in a $|0\rangle$ state, it would remain in a $|0\rangle$ state upon measurement.

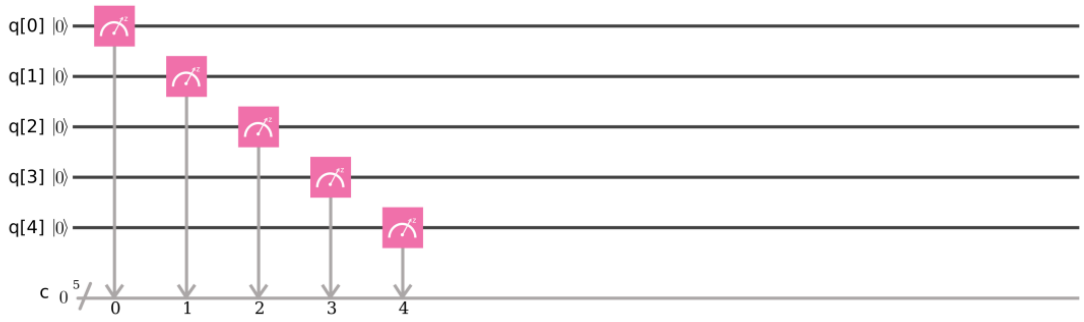

Figure 3.9. Experiment 3.1: Measuring the initial state.

When the circuit from Figure 3.9 was run on the *ideal simulator*, the results were as expected, measuring `00000` for each of the 1000 instances the simulation was completed. Each qubit was measured in the same $|0\rangle$ state from when it was initialized. The results of this experiment are found in Figure 3.10. The same circuit was also run on the *ibmqx2* on 20 April 2017, and the results are also shown in Figure 3.10. Of the 1024 shots on the *ibmqx2*, 996 were measured with the correct state of the system being `00000`. This corresponds to an overall error rate of 2.73%. With each experiment on the *ibmqx2*, IBM provides the results from the most recent calibration. The readout error provided by included the following values in IBM Experiment 3.1:

- q_0- 2.2×10^{-2}
- q_1- 3.2×10^{-2}
- q_2- 3.2×10^{-2}
- q_3- 5.0×10^{-2}

- q_4- 2.2×10^{-2}

The average readout error rate across all five qubits was 3.16×10^{-2}. The experimental results slightly outperform the reported error rate from the provided calibration results. There were no instances where more than one qubit at at time was measured in a $|1\rangle$ state. From this simple experiment it is easy to see not only how susceptible quantum computers are to errors, but also how it is possible to easily verify the correct final state. A scan of the results shown in Figure 3.10 makes it clear which result has the greatest probability of being the correct answer, and checking those results classically can be accomplished in a reasonable time-frame.

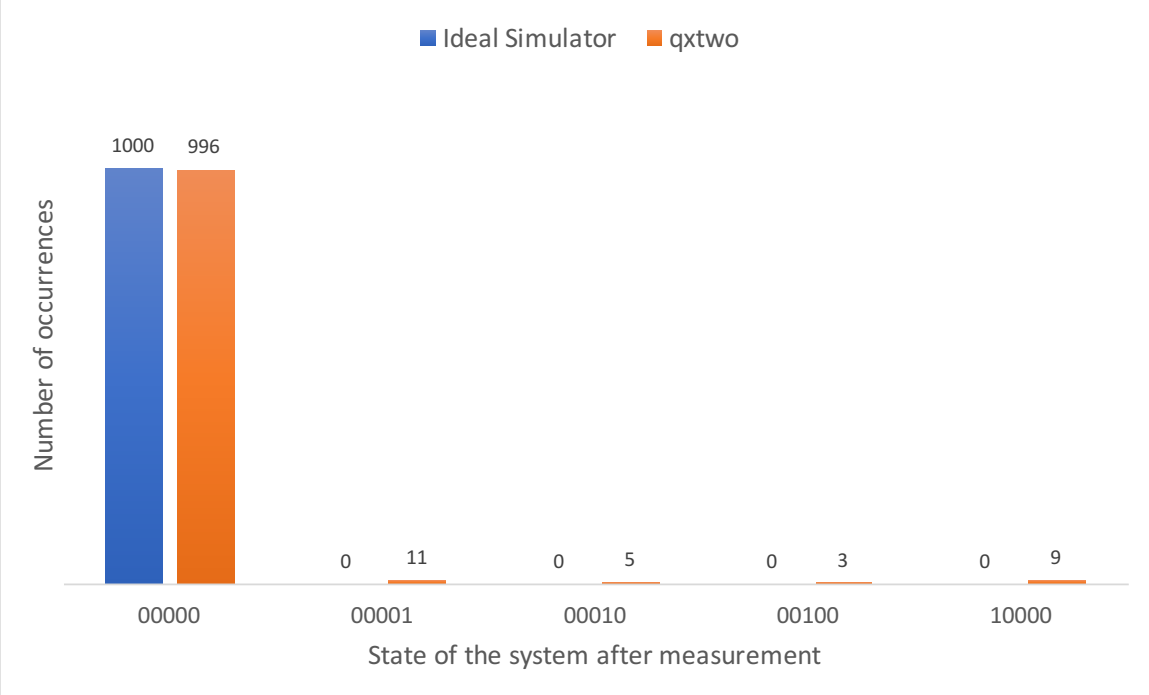

Figure 3.10. Results from Experiment 3.1: Measuring the initial state of the system with no additional gates applied.

Experiment 3.2: Qubit flips

The next test applies a single rotation gate to each qubit. This is accomplished by applying an $\boldsymbol{X}$ gate to each of the qubits and then taking a measurement of the each qubit, as shown in Figure 3.11.

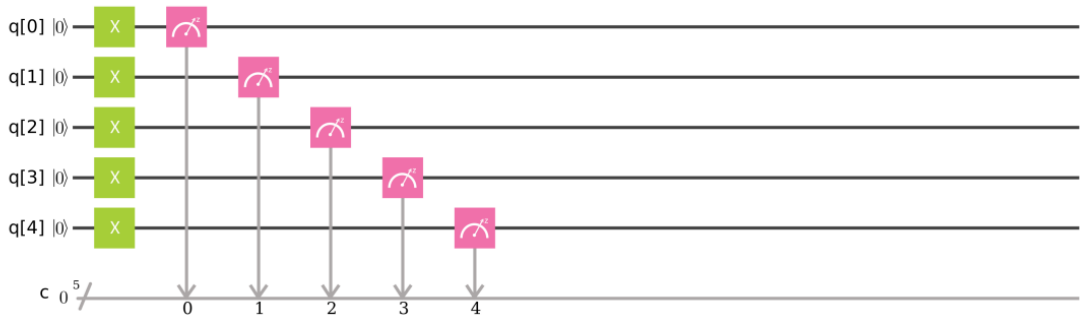

Figure 3.11. Experiment 3.2: Testing qubit flips with an ***X*** gate on q_0- q_4.

The expected result for this experiment would be that all qubits are in a $|1\rangle$ state at the final measurement. The results of this experiment when performed on the simulator indeed match the expectation. The results from running this circuit on the *ideal simulator* and the *ibmqx2* can be seen in Figure 3.12. The results show a decrease in the number of times the correct state was measured from Experiment 3.1. The experimental error rate in this instance was was 22.5%.

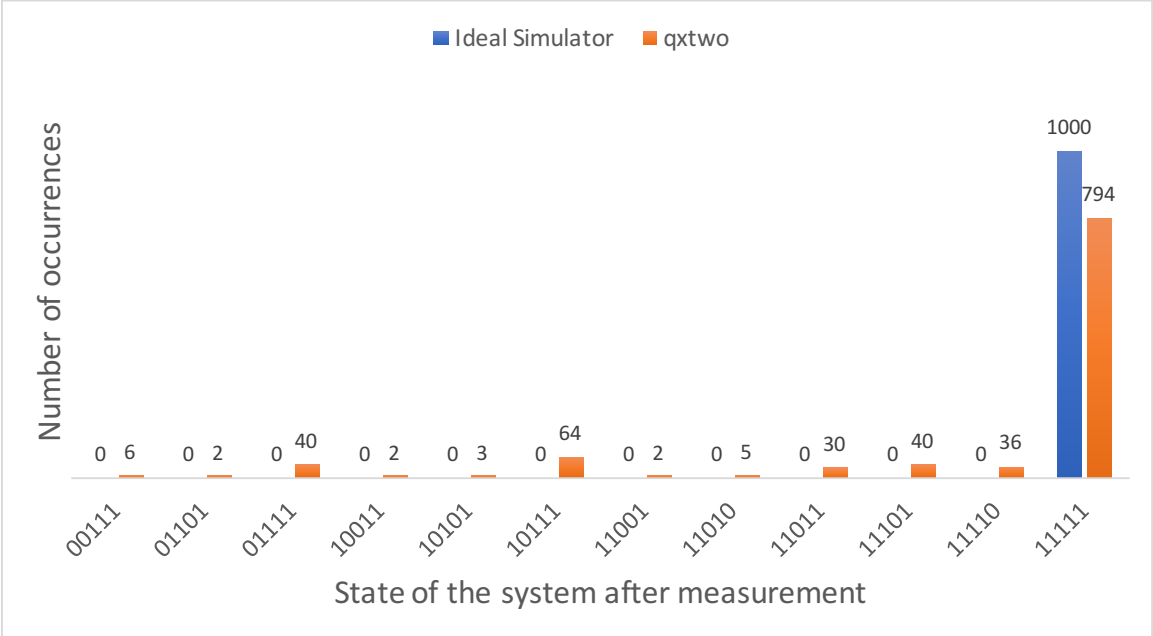

Figure 3.12. Results from Experiment 3.2: Testing qubit flips with an ***X*** gate on q_0- q_4.

The results from Experiment 3.2 on the *ibmqx2* again show a limited number of errors, but they have increased from the number of errors found in Experiment 3.1 on the *ibmqx2*. The ***X*** gate, although an abstraction of the physical gates implemented by IBM, has a specific error rate for each qubit. These error rates are provided in the results from IBM; the last time a calibration was completed on the *ibmqx2* showed a reasonably small error rate[5]. There were no instances of more than three qubits being measured in the incorrect state of $|0\rangle$ in a single run. Furthermore, of the instances where an incorrect state was measured, those with only one qubit in a $|0\rangle$ state were significantly more likely than those with two qubits in an incorrect state.

3.4.1 Superposition

Experiment 3.3: q_0 in Superposition with a Single Measurement
This next test on the *ibmqx2* will demonstrate the property of superposition by applying an ***H*** gate to q_0 and then taking a measurement. The score for placing q_0 into superposition is shown in Figure 3.13.

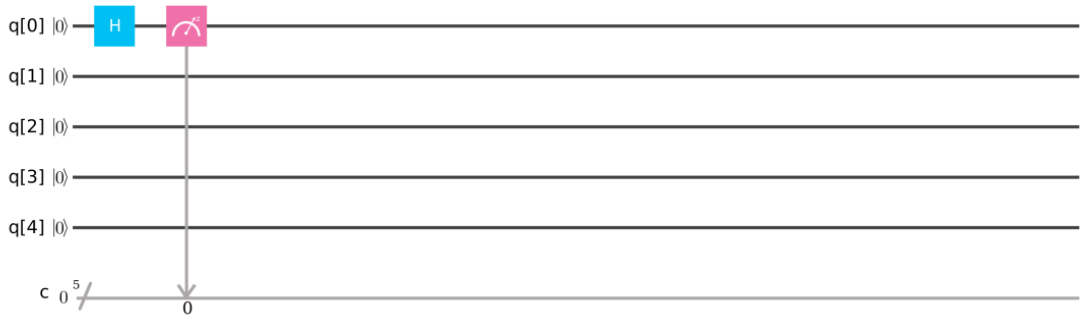

Figure 3.13. Experiment 3.3: Testing superposition with q_0 with a single measurement of the system.

A single measurement is added to this experiment. The expected result in an ideal situation would find 50% of the measurements of q_0 in a $|1\rangle$ state and 50% of the measurements q_0

[5]The gate error rate on the *ibmqx2* from the calibration performed on April 13th, 2018 at 08:53:48 for q_0 was $1.37\text{x}10^{-3}$.

in a $|0\rangle$ state.

The results in Figure 3.14 are close to an even distribution of q_0 being in a $|0\rangle$ or $|1\rangle$ state, as predicted. The results when Experiement 3.3 was run on the *ibmqx2* were similar to those on the *ideal simulator* and are shown in Figure 3.14.

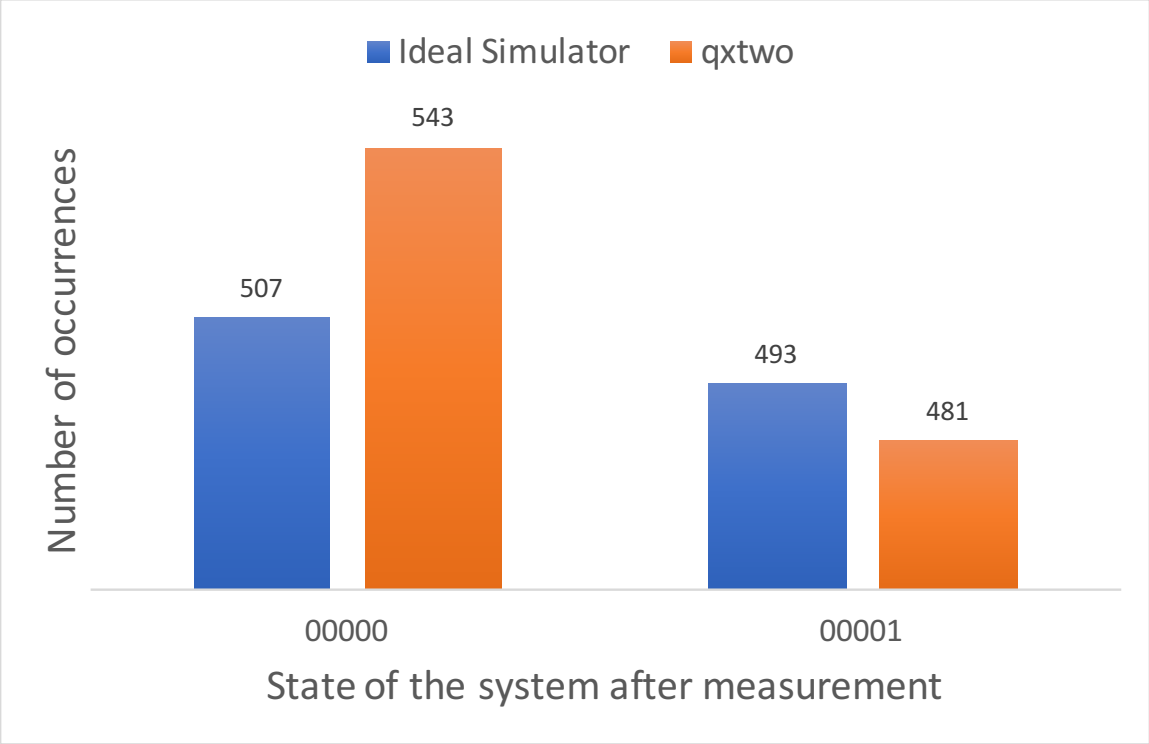

Figure 3.14. Results from Experiment 3.3: Testing superposition with q_0 and a single measurement of the system.

One reason for the limited noise seen in the results from Figure 3.14, specifically when implemented on the *ibmqx2*, is the single measurement. The lack of read errors resulting from the measurements of q_1 through q_4 indicates that they were not measured and assumed to remain in a $|0\rangle$ state. The actual state of the system will have similar variation, as seen in the first experiment, where a measurement of the system was taken with no additional gates applied.

Experiment 3.4: q_0 in Superposition with Five Measurements

To demonstrate the effect multiple measurements have on the results, we modify Experiment 3.3. Starting with the score from Figure 3.13, additional measurements are added to the remaining qubits q_1 through q_4, as shown in Figure 3.15.

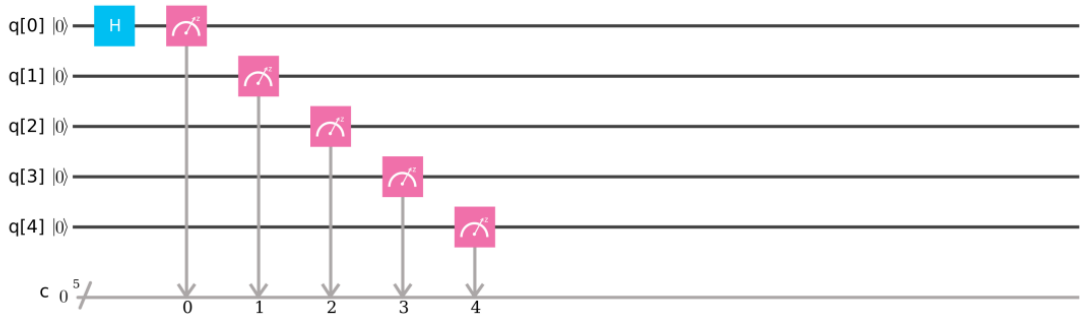

Figure 3.15. Experiment 3.4: q_0 into superposition with five measurements.

Experiment 3.4 is run on the *ideal simulator* to validate the hypothesis. The results from Experiment 3.4 on the *ideal simulator* should be consistent with those from Experiment 3.3 on the *ideal simulator*. Since no artificial noise is injected into the *ideal simulator*, the additional measurement gates should not find any unexpected states. The results from Experiment 3.4 on the *ideal simulator*, shown in Figure 3.16, are nearly identical to both the expected results and the results of Experiment 3.3 on the *ideal simulator*. The results show an even distribution of the the final states `00000` and `00001`, with no error states being computed.

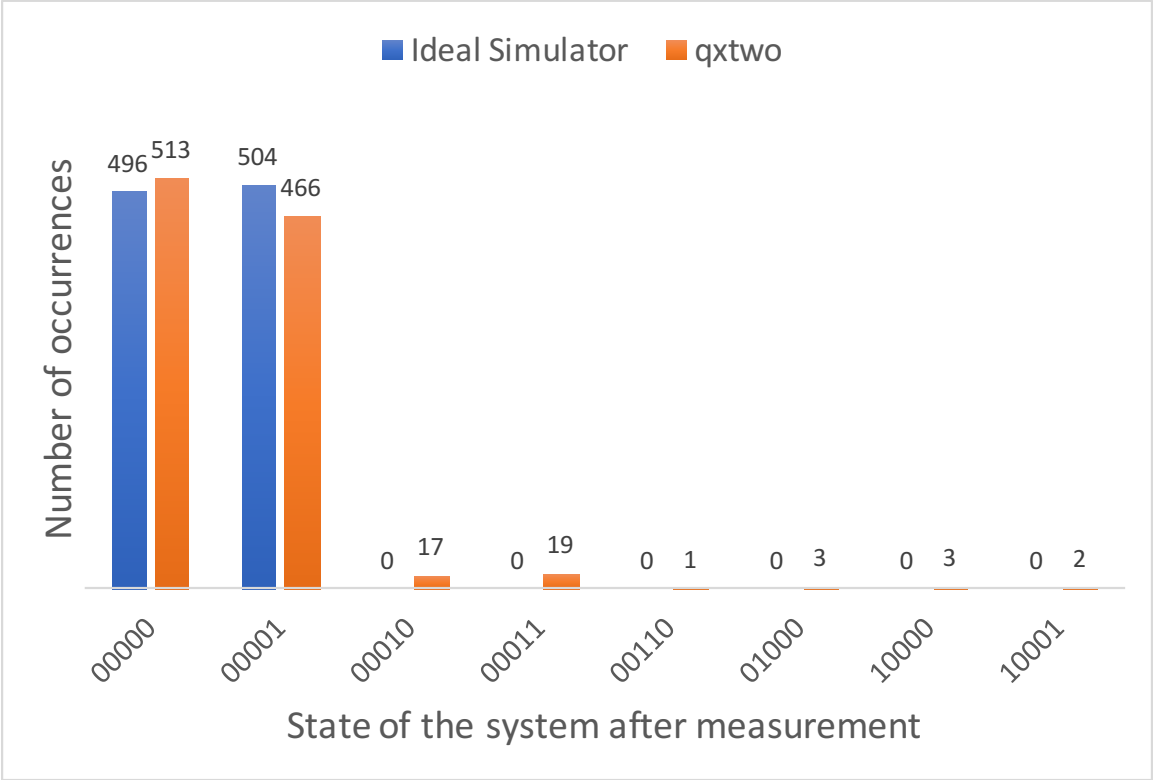

Figure 3.16. Results from Experiment 3.4: q_0 in superposition with five measurements.

The results from running Experiment 3.4 on the *ibmqx2* show expected error levels which were absent in Experiment 3.3. The results in Figure 3.16 show the additional noise not present when the only measurement of the system was taken on q_0. The results shown in Figure 3.16 reflect the full state of the system after q_0 is placed into superposition. Although the noise shown in this experiment is small, and the distribution between the correct states `00000` and `00001` is still within our expectations, it is a significant deviation from classical computing. You must account for noise in a quantum computer.

Experiment 3.5: Five Qubits in Superposition with Five Measurements

A similar experiment can be performed by placing all five qubits into superposition and then taking a measurement. The expected results are an even distribution of all 2^N states of the system, where N is the number of qubits of the system. On the *ibmqx2*, this would be 2^5 (32) states, ranging from $|00000\rangle$ to $|11111\rangle$. Figure 3.17 shows the quantum score used

to create this state.

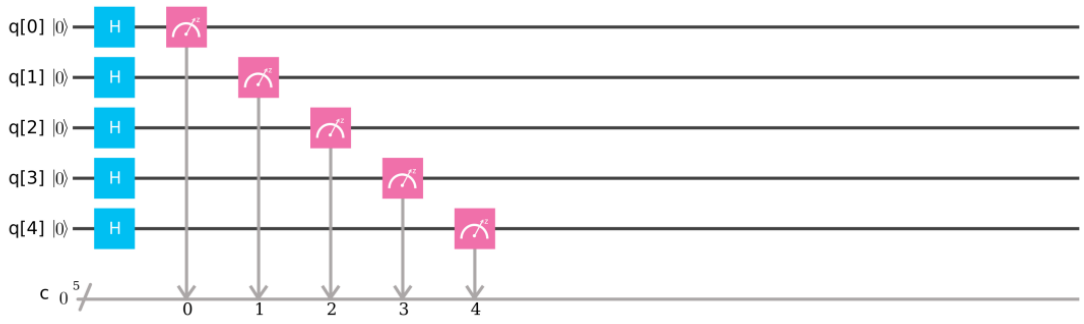

Figure 3.17. Experiment 3.5: Five qubits in superposition.

Running Experiment 5 a total of 1000 times on the *ideal simulator*, a perfect result would measure each state 31.25 times. The observed results from testing Experiment 3.5 on the *ideal simulator* are Figure 3.18. The results show an even distribution of occurrences of the 32 possible final states. The results on both the *ideal simulator* and the *ibmqx2* both appear to have a normal distribution about the expected mean.

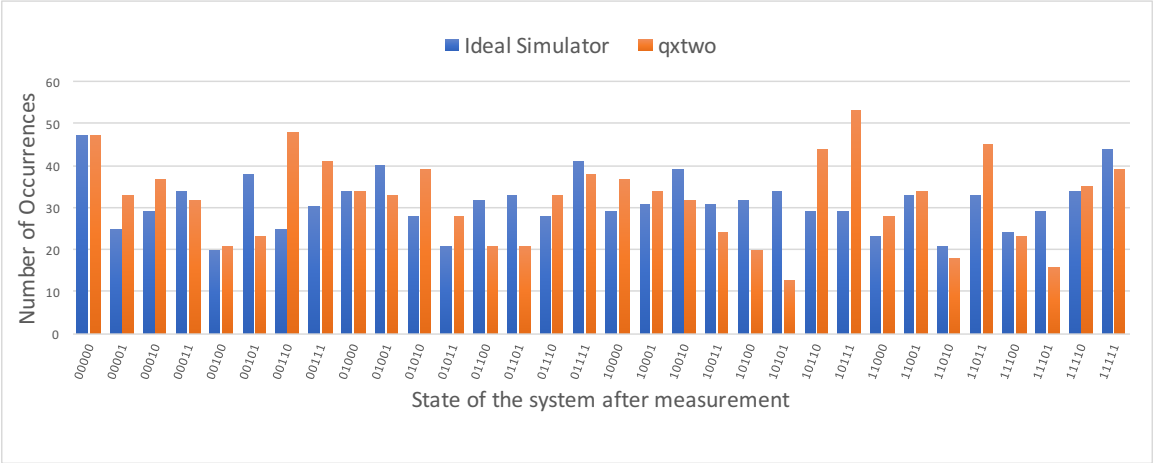

Figure 3.18. Results from Experiment 3.5: Five qubits in superposition with five measurements.

Experiment 3.5a: Testing Superposition with Additional Shots

As mentioned at the start of this section, the previous experiments have used the default shot number (1024) for the *ibmqx2* and 1000 shots for the *ideal simulator* to make the results comparable. The *IBM Quantum Experience* allows users to select two additional settings for the real processors, 4096 and 8192 shots, and the shots for the *ideal simulator* are user-defined. Increasing the number of shots for an experiment, either real or simulated, should move the distribution of results closer to the expected values. Figure 3.20 shows the results from three separate instances of running the experiment from Figure 3.19 on the simulator. Experiment 3.5a uses the same score as Experiment 3.5.

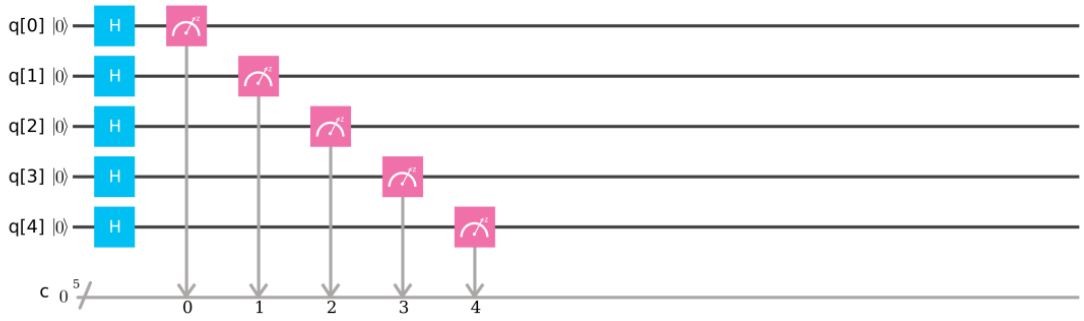

Figure 3.19. Experiment 3.5a: Testing superposition with additional shots.

The only difference expected in running Experiment 3.5 three separate times would be a change in the number times each state is observed with the relative distribution of those states remaining constant. The probability of each state should remain at 3.125% of the number of shots. Figure 3.20 shows the results of running the experiment with the shot parameter set to 1000, 4000, and 8000. The results scale with the increased number of shots when this experiment is executed on the *ideal simulator*.

Figure 3.20. Results from Experiment 3.5a with 1000, 4000, and 8000 shots on *ideal simulator*.

Figure 3.21 shows the results for the same quantum score, run on the *ibmqx2* with 1024, 4096, and 8192 shots. The results from Experiment 3.5a with 1024 and 4096 shots appear to have a similar distribution of results as shown in simulation. When the results of Experiment 3.5a with 8192 shots are examined there emerges a clear pattern. This same pattern can be found in the other iterations on the *ibmqx2*, but with fewer shots, the error patterns are more difficult to distinguish from a random distribution.

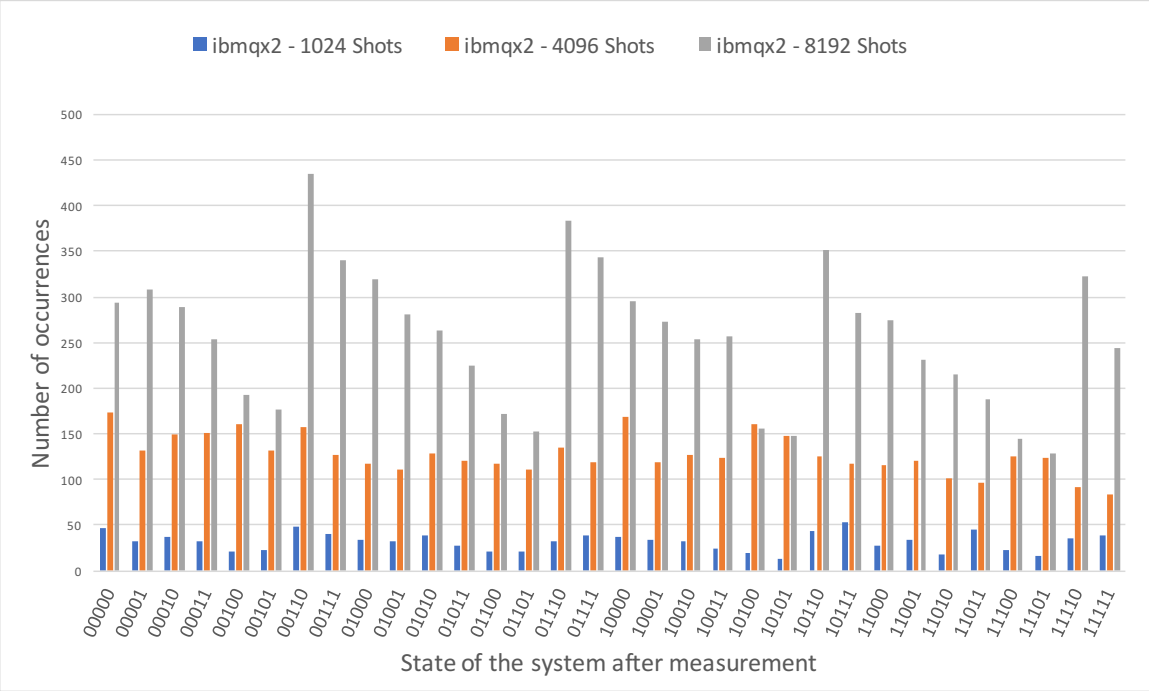

Figure 3.21. Results from Experiment 3.5a with 1024, 4096, and 8192 shots on *ibmqx2*.

3.4.2 Entanglement

Experiment 3.6: Creating *Bell* State $|\beta_{00}\rangle$

The final experiment in this section is to demonstrate entanglement. We will create the *Bell* state $|\beta_{00}\rangle$ from Figure 2.8. The quantum score in Figure 3.22 creates this entangled state between q_0 and q_1.

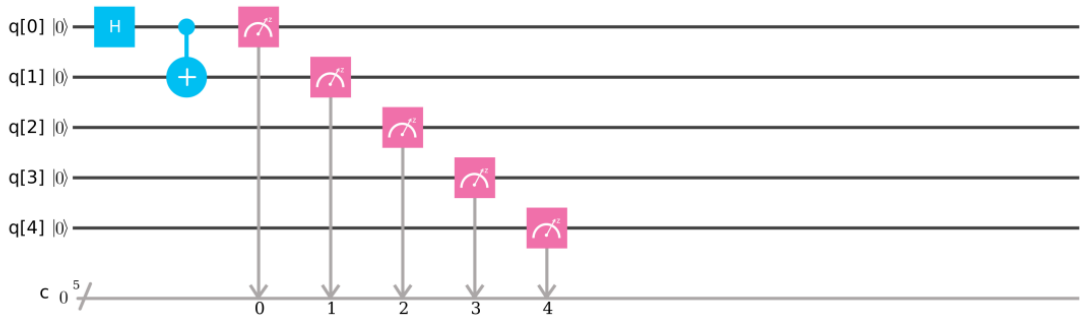

Figure 3.22. Experiment 3.6: Creating *Bell* state $|\beta_{00}\rangle$.

When creating the *Bell* state for Experiment 3.6, we expect that there will be an even distribution of the states `00000` and `00011`. The results from Experiment 3.6 are shown in Figure 3.23. The results from the *ideal simulator* and *ibmqx2* show an even distribution of the correct states, as expected. The error rates shown are consistent with previous experiments.

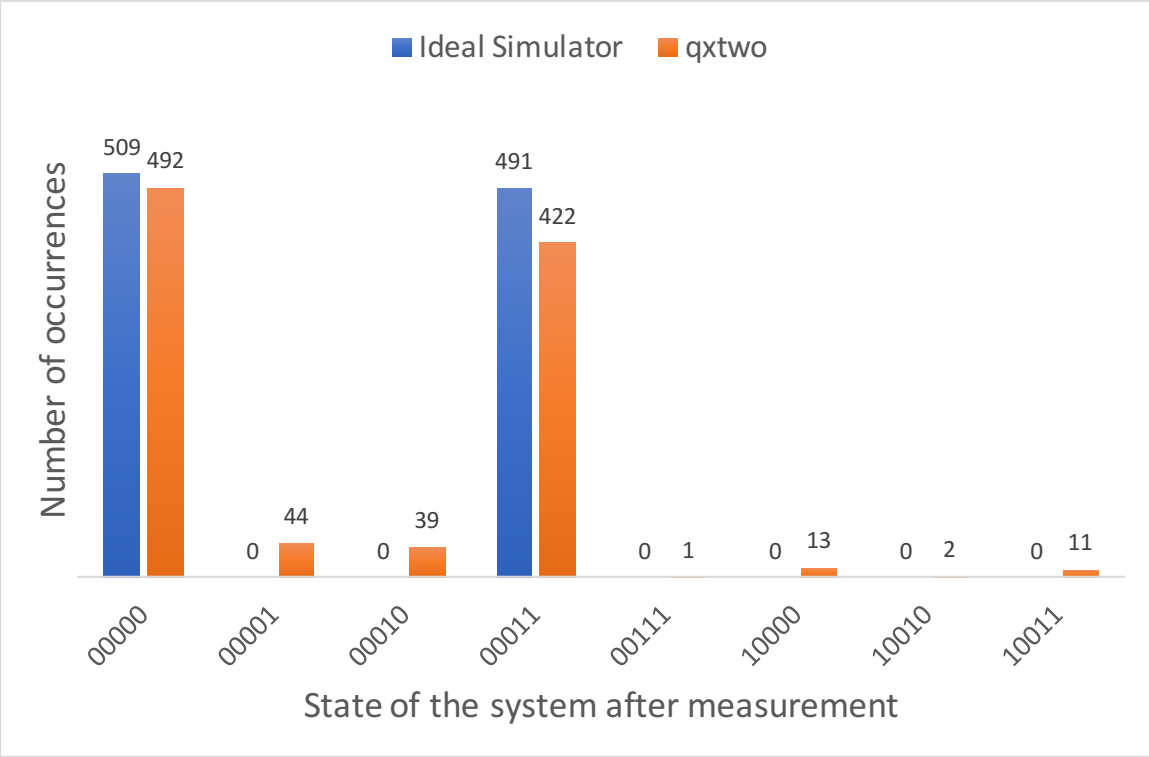

Figure 3.23. Results from Experiment 3.6: Creating *Bell* state $|\beta_{00}\rangle$.

CHAPTER 4:
Experimenting with Quantum Algorithms

4.1 Deutsch-Jozsa's Algorithm

The first algorithm to test on the *ibmqx2* is the Deutsch-Jozsa algorithm. The algorithm is admittedly a contrived example, but does demonstrate the quantum speed-up that the algorithm achieves over any known classical algorithm. Deutsch-Jozsa's algorithm determines whether a function ***f(x)*** is balanced or constant. Balanced means that exactly half of the inputs ***x*** map to ***f(x)*=0**, and the other half map to ***f(x)*=1**; constant meaning that all inputs map to either 0 or 1 [1], [4], [18]. Figures 4.1 and 4.2 show examples of balanced and constant functions, respectively. In Deutsch-Jozsa's algorithm, it is guaranteed that the function will be constant or balanced.

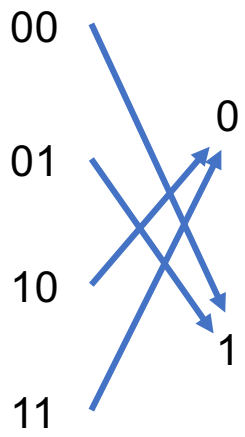

Figure 4.1. Example Deutsch-Jozsa balanced function. Adapted from [18].

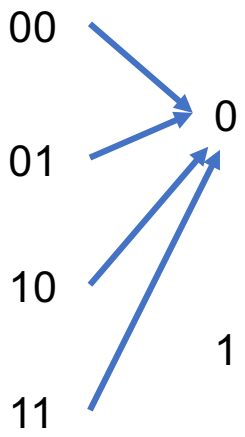

Figure 4.2. Example Deutsch-Jozsa constant function.

With a classical algorithm, the worst case would require checking $2^{n-1} + 1$ inputs to know that the function is constant. Deutsch-Jozsa's algorithm can outperform these classical algorithms through the use of an oracle. The steps described by *Yanofsky* and *Nielsen and*

Chuang for Deutsch-Jozsa's can be broken down into five discrete steps.

1. Initialize the state of the system to $|0\rangle^{\otimes n}|1\rangle$.
2. Apply $\boldsymbol{H}$ gates to the system to place it into superposition.
3. Apply U_f.
4. Reapply $\boldsymbol{H}$ gates to the system.
5. Measure the state of the system. [1], [18].

The matrix that corresponds to Figure 4.1 is shown in Equation 4.1.

$$\begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.1}$$

IBM describes the steps almost identically, but the first step does not include initializing the last qubit as $|1\rangle$. The first step for IBM is shown below. The remaining four states remain consistent with those described in [1], [18].

1. Initialize the state of the system to $|0\rangle^{\otimes n}$. [10]

All three sources agree that if f is constant, the probability is one that the final measured state will be 0^n, and if f is balanced, the probability is zero that the final state will be 0^n. Therefore, a single evaluation of the function will determine whether it is constant or balanced. IBM presents two examples in the *User Guide* of Deutsch-Jozsa's algorithm, one example of a constant function and one example of a balanced function.

4.1.1 Experiment 4.1: Deutsch-Jozsa Constant Function

The example of a constant function provided by IBM is essentially $f(x) = x$; in this case, the U_f is described in Equation 4.2. Although no gates are applied to the qubits between the

set of $\boldsymbol{H}$ gates, an implicit $\boldsymbol{I}$ gate is applied to each qubit, resulting in the matrix in Equation 4.2.

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.2}$$

The quantum score used for the constant function example from IBM is shown in Figure 4.3. The example here is simplistic, but it does follow the five steps described previously. The first step was to initialize the starting state of the system. The default initial state of *ibmqx2* is $|00000\rangle$, so no additional action is necessary to complete step one. The three $\boldsymbol{H}$ gates place the system into superposition, as required by step two. U_f is implicitly applied to the system after the first set of $\boldsymbol{H}$ gates[6]. This fulfills step three. Three $\boldsymbol{H}$ gates are again applied to the system fulfilling step four of the process. The fifth and final step is applying measurement gates to q_0, q_1, and q_2.

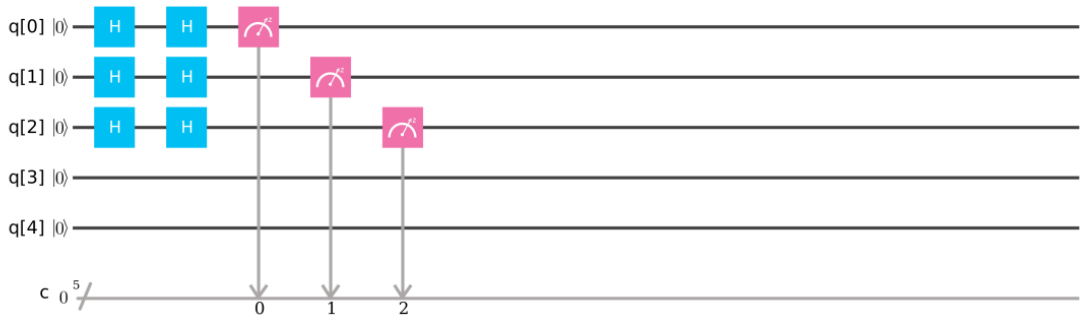

Figure 4.3. Experiment 4.1: Deutsch-Jozsa constant equation example. Source: [10].

[6]Not applying a gate is equivalent to applying the $\boldsymbol{I}$

This algorithm is simple enough that the expected result can be determined prior to running it in simulation or on the *ibmqx2*. Chapter 1 provided the necessary background to compute $\boldsymbol{H} \otimes \boldsymbol{H} \otimes \boldsymbol{H}$; the result is shown in Equation 4.3.

$$\boldsymbol{H} \otimes \boldsymbol{H} \otimes \boldsymbol{H} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \end{bmatrix} \tag{4.3}$$

The $\boldsymbol{H}$ gate has several interesting properties, one of which is that the $\boldsymbol{H}$ is its own inverse. This property is shown in Equation 4.4. This can be repeated with the 8-by-8 $\boldsymbol{H}$ matrix, where $(\boldsymbol{H} \otimes \boldsymbol{H} \otimes \boldsymbol{H}).(\boldsymbol{H} \otimes \boldsymbol{H} \otimes \boldsymbol{H})$ results in the matrix from Equation 4.3.

$$\begin{aligned} \boldsymbol{H}.\boldsymbol{H} &= \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{-1}{\sqrt{2}} \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{-1}{\sqrt{2}} \end{bmatrix} \\ &= \begin{bmatrix} \frac{1}{2} + \frac{1}{2} & \frac{1}{2} - \frac{1}{2} \\ \frac{1}{2} - \frac{1}{2} & \frac{1}{2} + \frac{1}{2} \end{bmatrix} \\ &= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \\ &= \boldsymbol{I} \end{aligned} \tag{4.4}$$

The quantum score from Figure 4.3 effectively does nothing. Therefore, after execution, the initial state, $|000\rangle$, should be the final measured state. The results from running Experiment 4.1 on the *ideal simulator* are shown in Figure 4.4, and are consistent with the *Mathematica* calculations shown in Appendix B. The results of this simulation support the hypothesis which was determined previously, that the function, U_f, defined by Equation 4.2, is constant.

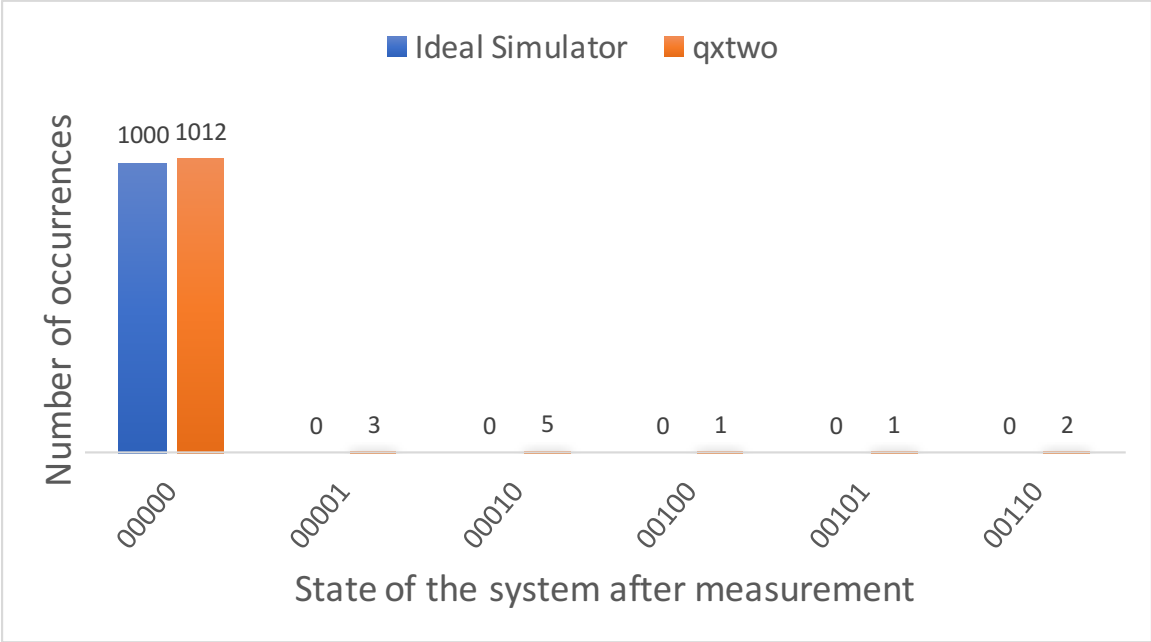

Figure 4.4. Results from Experiment 4.1: Deutsch-Jozsa's constant equation example.

This experiment was also run on the *ibmqx2*; the results are shown in Figure 4.4. Theoretically, this experiment only needs to evaluate U_f once to determine whether function f is constant. The error rate for this circuit is comparable to Experiment 3.1, where all five qubits were measured with no other gates applied.

This could be an indication of the way in which the quantum scores are compiled prior to running on a quantum processor. If the $\boldsymbol{H}$ gates were applied to each of the three qubits two separate times as shown in the quantum score for this circuit, there should be an increase in the error rate observed relative to Experiment 3.1. If, however, the pre-processing of this circuit determined that it is functionally equivalent to Figure 4.5, it may have compiled it as such prior to execution. The available documentation on the *QASM* compiler process does not have sufficient information to determine the validity of this claim, but the experimental results do support this hypothesis.

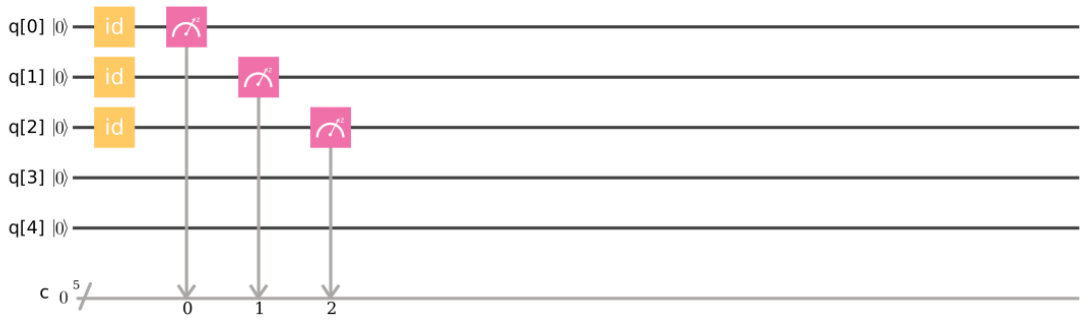

Figure 4.5. An equivalent quantum score to Deutsch-Jozsa constant function example from IBM.

4.1.2 Experiment 4.2: Deutsch-Josza Balanced Function

The example circuit for a balanced function is shown in Figure 4.6. The U_f in this score is between the two sets of $\boldsymbol{H}$ gates. This function is $f(x) = x_0 \oplus x_1 x_2$ [10]. Equation 4.5 was computed using *Mathematica* to show the matrix which implements this function, the *Mathematica* for computing is found in Appendix B. The expectation is that the result will never be $|000\rangle$.

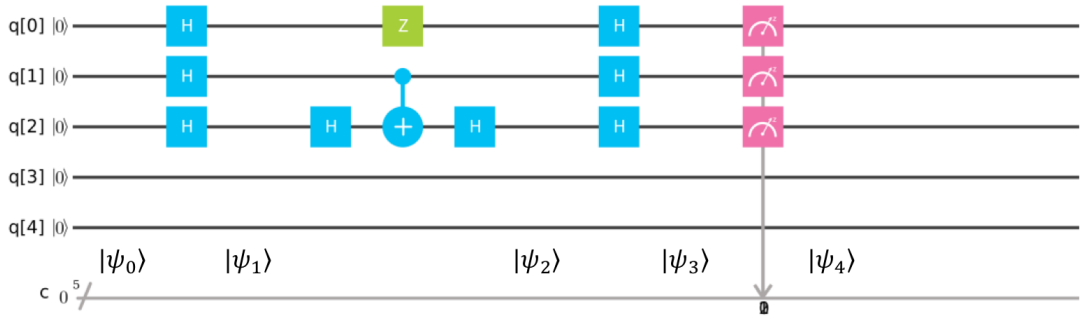

Figure 4.6. Experiment 4.2: Deutsch-Jozsa's balanced equation example. Adapted from [10].

$$U_f = (\boldsymbol{I} \otimes \boldsymbol{I} \otimes \boldsymbol{H}).(\boldsymbol{Z} \otimes \boldsymbol{CNOT}).(\boldsymbol{I} \otimes \boldsymbol{I} \otimes \boldsymbol{H}) = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.5}$$

As a requirement of the Deutsch-Jozsa algorithm, this function is required to be either balanced or constant. We can check the function classically, which will take at most $2^{n-1} + 1$ inputs, which would be five in this example. By starting with the state $|000\rangle$ and incrementing through the possibilities, we find for state $|011\rangle$ that $f(x) = 1$, as shown in Equation 4.6. We know that the function is indeed balanced, but it took four evaluations of $f(x)$ to make that determination. Deutsch-Jozsa will yield the same conclusion in a single evaluation of U_f.

$$\begin{aligned} f(000) &= 0 \oplus 0 \wedge 0 = 0 \\ f(001) &= 0 \oplus 0 \wedge 1 = 0 \\ f(010) &= 0 \oplus 1 \wedge 0 = 0 \\ f(011) &= 0 \oplus 1 \wedge 1 = 1 \\ f(100) &= 1 \oplus 0 \wedge 0 = 1 \\ f(101) &= 1 \oplus 0 \wedge 1 = 1 \\ f(110) &= 1 \oplus 1 \wedge 0 = 1 \\ f(111) &= 1 \oplus 1 \wedge 1 = 0 \end{aligned} \tag{4.6}$$

The quantum circuit provided by IBM, shown in Figure 4.6, follows the same five steps as described previously. The system is initialized by default to the state $|000\rangle$ at $|\psi_0\rangle$. At state $|\psi_1\rangle$, the system is in a state of superposition. Once the oracle U_f has been applied, the

system is in state $|\psi_2\rangle$. Again, the $\boldsymbol{H}$ gates are applied, resulting in state $|\psi_3\rangle$, followed by the measurement, which results in the final state $|\psi_4\rangle$.

The results from simulation and real-world execution of this circuit on the *ibmqx2* are found in Figures 4.7. Both sets of results do not show, as expected, the state $|000\rangle$. They show an equal distribution of states $|100\rangle$, $|101\rangle$, $|110\rangle$ and $|111\rangle$[7]. Since this circuit, excluding the errors, does not return the state $|000\rangle$, a single evaluation of the function has determined that it is balanced. As with the constant function example of Deutsch-Jozsa's algorithm, we have demonstrated a speed-up over a classical system, which would have required as many as $2^{n-1} + 1$ function evaluations.

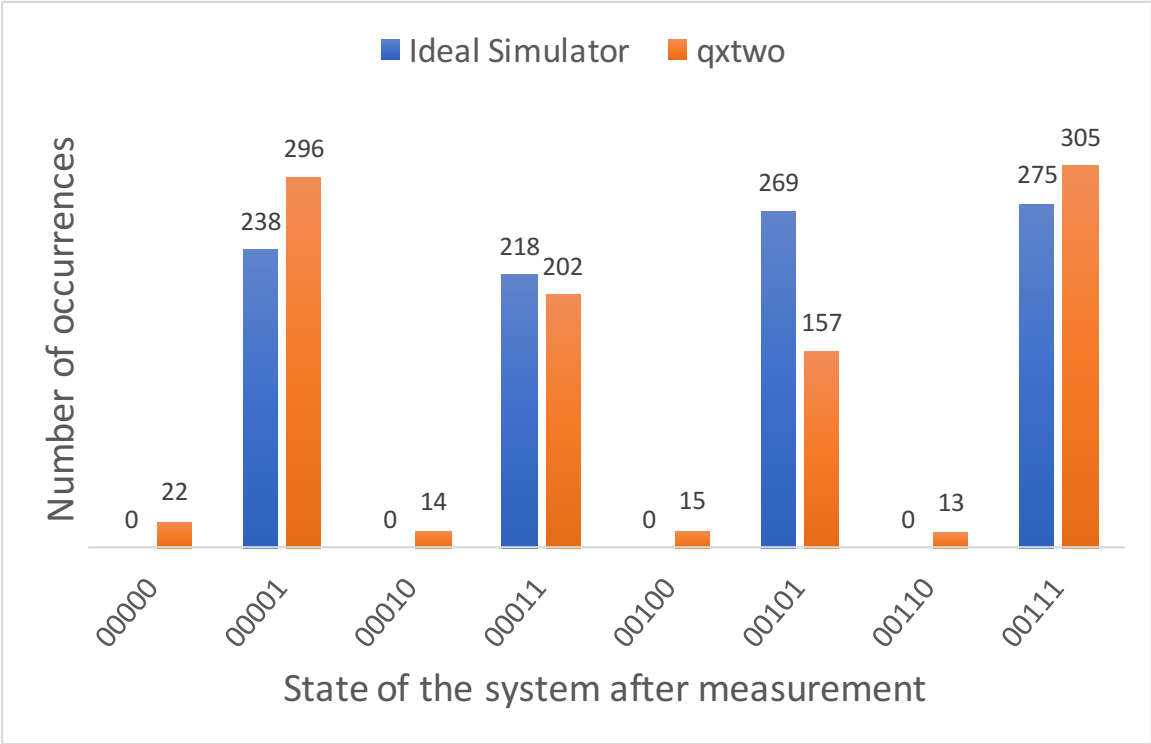

Figure 4.7. Results from Experiment 4.2: Deutsch-Jozsa's balanced equation example.

[7]The default output of from the IBM quantum processors inverts the bits after measurement. The state $|100\rangle$ is therefore represented in the classical register as `00001`.

4.2 Grover's Algorithm

Although Deutsch-Jozsa's algorithm provides an exponential speed-up over classical algorithms, it is a purposefully contrived example to demonstrate this possibility. The range of applications of their algorithm is limited. Grover's search algorithm achieves a quantum speed-up of for search and several related problems. Decreasing storage costs and the proliferation of devices that generate large amounts of data, i.e. IOT devices, have motivated the emerging field of Big Data. CERN reported a record level of data stored in October of 2017 with 12.3 petabytes [35]. Searching through data at that scale is a computationally costly.

We can frame the problem by describing an array of length N that stores unsorted data, as shown in Figure 4.8. In the worst case, it would take N array accesses to find the data needed, $\frac{N}{2}$ on average. Grover's algorithm will accomplish the same task in $O\sqrt{N}$ queries [5].

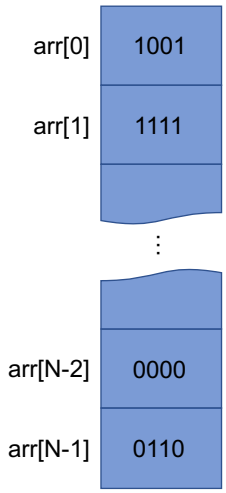

Figure 4.8. An unsorted array of length N.

Grover's algorithm uses the amplitude amplification technique to achieve quadratic speed up over classical search [10]. This technique can be broken down into two distinct processes: phase inversion and inversion about the mean [18]. Grover's algorithm starts by placing the system into a state of superposition by applying $\boldsymbol{H}^{\otimes n}$. The oracle U_f is applied to invert the phase of the answer sought. The state of the system is then inverted about the mean. Step two is then repeated $O\sqrt{N}$ times, and then the system is measured. To understand this algorithm, a geometric explanation based on vectors is helpful. Figure 4.9 shows how this

process increases the probability amplitudes of the desired answer.

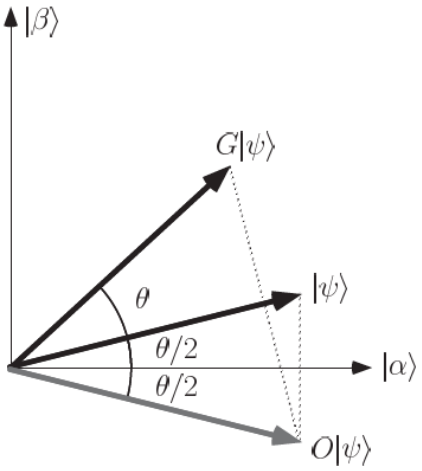

Figure 4.9. Effects of Grover's search algorithm on a vector. Source: [1].

There are four basic steps for Grover's algorithm:

1. Initialize the state of the system to $|0\rangle^{\otimes n}$.
2. Place the system into superposition.
3. Repeat $O\sqrt{N}$ times:
   Phase inversion.
   Inversion about the mean.
4. Measure the state of the system. [18]

These steps are shown in a Figure 4.10, which shows the quantum score for Grover's algorithm on a system of size n.

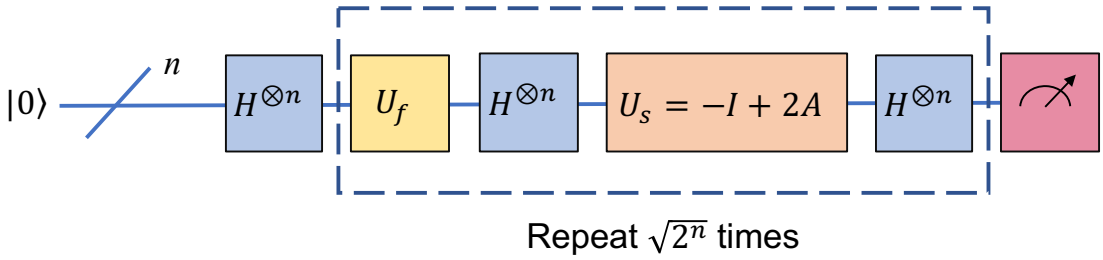

Figure 4.10. An arbitrary example of Grover's search algorithm. Adapted from [10], [18].

There are four possible examples of Grover's search algorithm when using two qubits. Each one looks for a different answer, `00`, `01`, `10`, and `11`, respectively. For each of the four possibilities, a new function U_f must be constructed to implement Grover's algorithm. Constructing the matrix for these functions is a tedious but trivial process. The process of creating that matrix from single qubit gates is not. This topic will be further discussed in Chapter 7. For now we will examine the functions as provided by IBM.

The four circuits are shown in Figures 4.11 through 4.14. The double ***H*** gates in each score provide a convenient comparison point for each implementation. As expected, U_f varies across the four circuits, but U_s remains constant. Though the construction of the functions is as expected, the circuits do not repeat phase inversion and inversion about the mean.

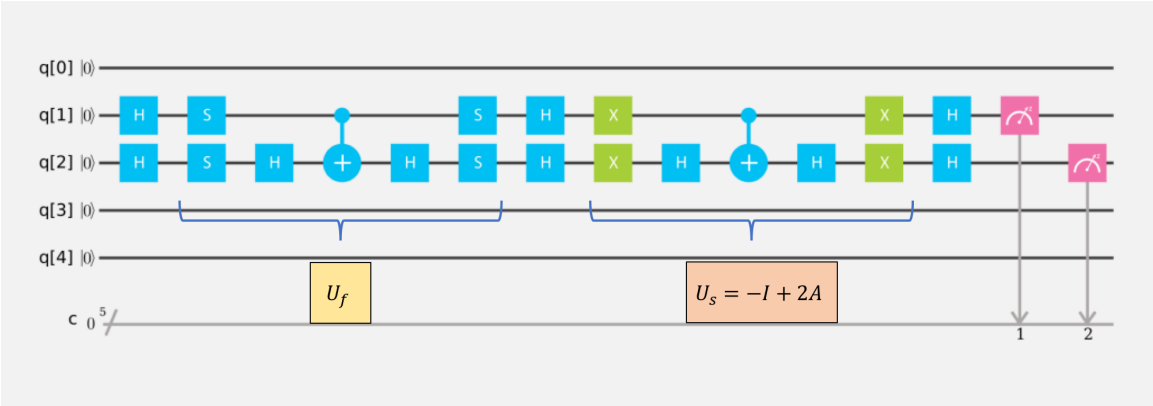

Figure 4.11. Experiment 4.3: Grover's search algorithm, where the desired answer is 00, and $N = 2$. Adapted from [10].

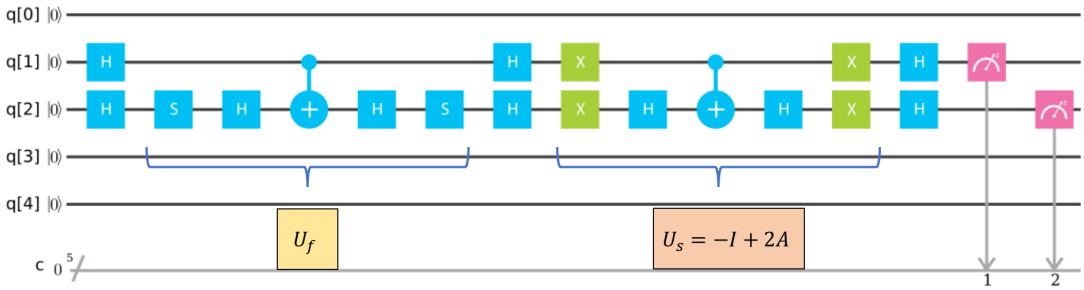

Figure 4.12. Experiment 4.4: Grover's search algorithm, where the desired answer is 01, and $N = 2$. Adapted from [10].

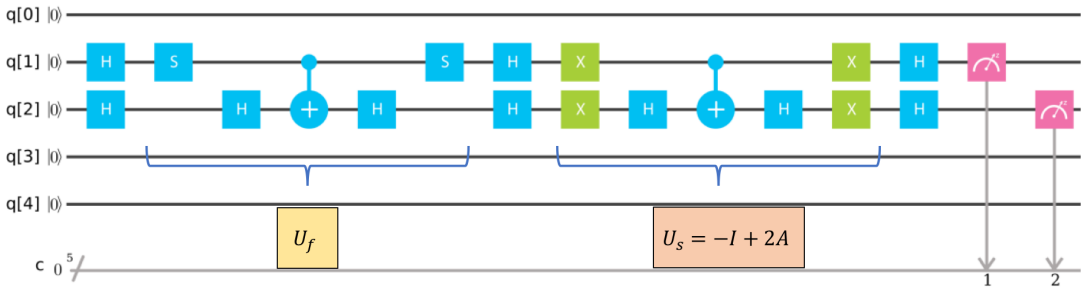

Figure 4.13. Experiment 4.5: Grover's search algorithm, where the desired answer is 10, and $N = 2$. Adapted from [10].

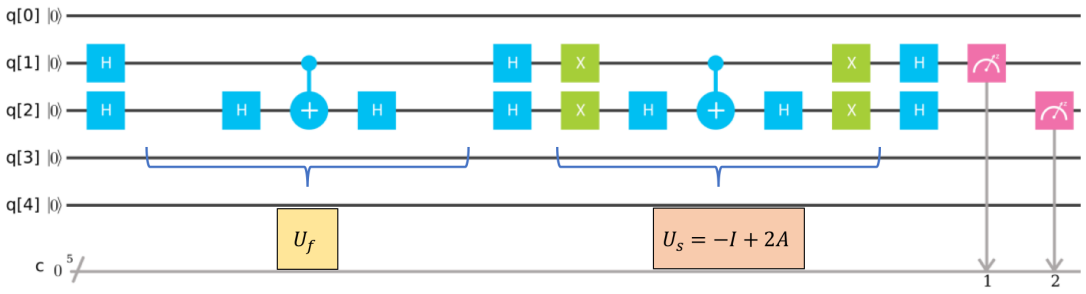

Figure 4.14. Experiment 4.6: Grover's search algorithm, where the desired answer is 11, and $N = 2$. Adapted from [10].

The results from each Experiments 4.3 - 4.6 are shown in Figures 4.15 - 4.18, respectively. The results from each experiment on the *ibmqx2* compared to the *ideal simulator* demonstrate a reasonable performance level of the system, with the exception of Experiment 4.4. Experiment 4.4 searches for `01`, which should be `00100`. The simulation of this experiment shows this result, but the results on the *ibmqx2* significantly deviate from this. The *Mathematica* notebooks for Experiments 4.3 - 4.5 are found in Appendix B.

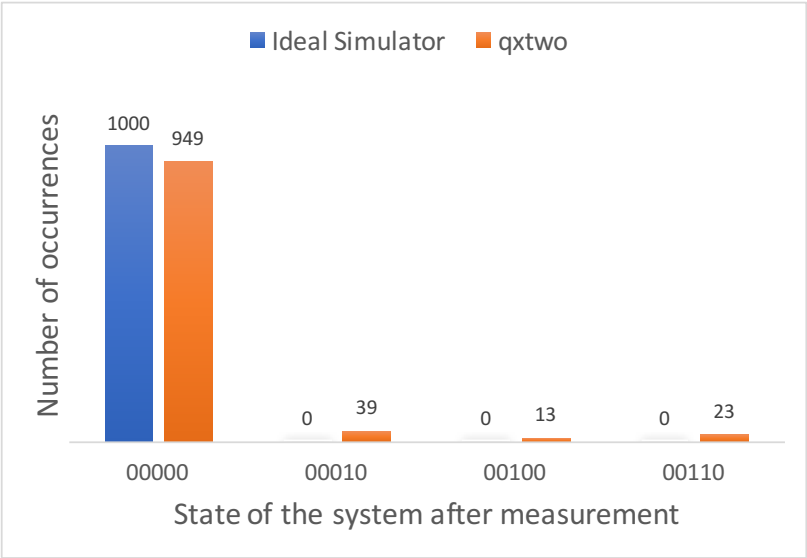

Figure 4.15. Results from Experiment 4.3: Grover's search algorithm, where the desired answer is 00, and $N = 2$.

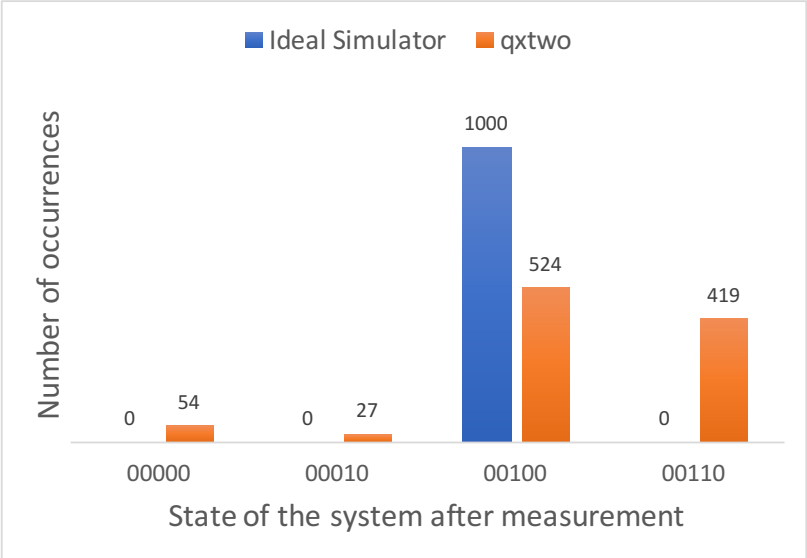

Figure 4.16. Results from Experiment 4.4: Grover's search algorithm, where the desired answer is 01, and $N = 2$.

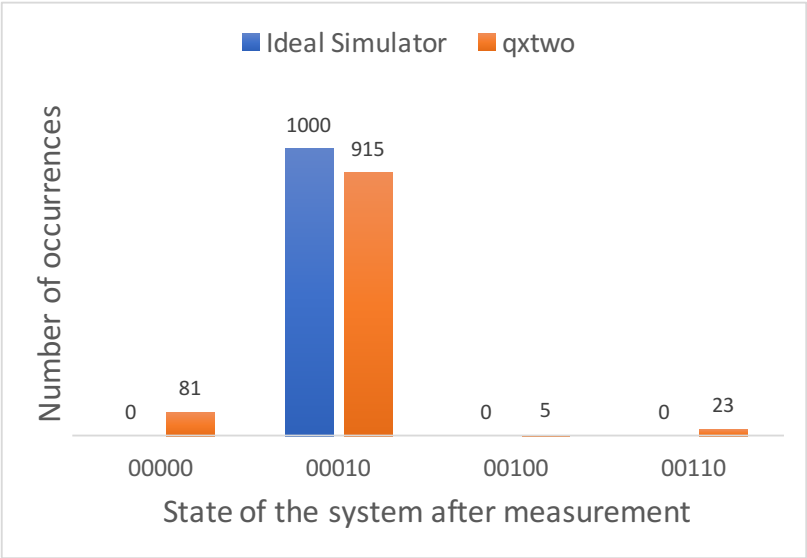

Figure 4.17. Results from Experiment 4.5: Grover's search algorithm, where the desired answer is 10, and $N = 2$.

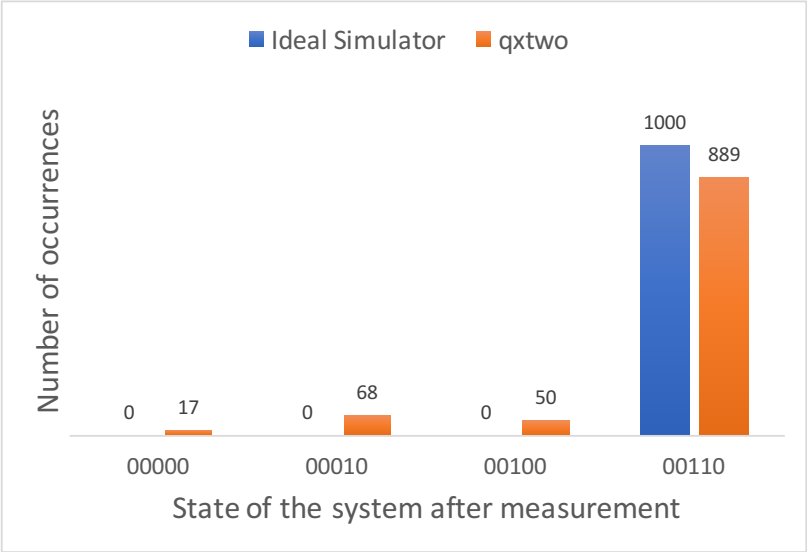

Figure 4.18. Results from Experiment 4.6: Grover's search algorithm, where the desired answer is 11, and $N = 2$.

4.3 Shor's Algorithm

Peter Shor's factoring algorithm presents an interesting case study for quantum algorithms. His algorithm is deceptively simple when distilled down into its discrete steps, but implementing each of those steps presents significant practical challenges. Yanofsky describes the steps of Shor's algorithm as such:

1. Determine if N is prime or a power of prime.
2. Randomly choose an integer a, where $1 < a < N$.
3. Use the circuit depicted in Figure 4.19 to find the period of N.
4. If r is odd, go back to step 2, where r is described in Equation 4.7.
5. Use Euclid's algorithm to compute greatest common denominator and return a non-trivial solution [18].

Of the steps described above, only the third step is performed on a quantum computer. In step three, Shor's algorithm takes the problem of integer factorization and changes it into the problem of period finding. As Shor states it, "the key idea of a quantum factoring algorithm is the use of a Fourier transform to find the period of the sequence $u_i = x^i(modN)$, from which period a factorization of N can be obtained" [36]. An example of how Shor's algorithm could be implemented is shown in Figure 4.19.

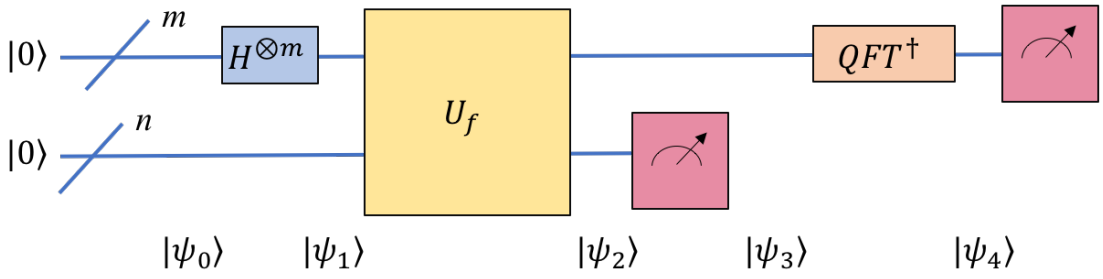

Figure 4.19. Shor's factoring algorithm. Adapted from [18].

Beauregard claims that you can implement a minimized version of Shor's algorithm needing only $2n+3$ qubits to factor an n-bit number [37]. This, however, leads to a trivial case when experimenting with a five-qubit system. There are two core elements of Shor's algorithm

that can be implemented directly on the *ibmqx2* directly: modular multiplication and the Quantum Fourier Transform (QFT).

IBM presents the topic of modular multiplication in the *Full User Guide* and provides the background as to how this is used in Shor's to find the period. The purpose of the modular multiplication here is to find the smallest possible r, such that:

$$a^r = 1(modN) \tag{4.7}$$

IBM provides the score shown in Figure 4.26 as an instance of Equation 4.7. In a full implementation of Shor's algorithm, you evaluate additional values of r. This score represents a part of U_f, from which the additional iterations could be implemented. The equation that the score in Figure 4.26 is solving is shown Equation 4.8. See Markov and Saeedi for the construction of their circuits for modular multiplication and exponentiation [10], [38]. IBM uses the same circuit construction as shown in [38] to build their modular multiplication gate. From [38] you can see that state $|\psi_0\rangle$ determines r, state $|\psi_1\rangle$ determines N, and state $|\psi_3\rangle$ determines a[8].

$$7^4 = 1(mod15) \tag{4.8}$$

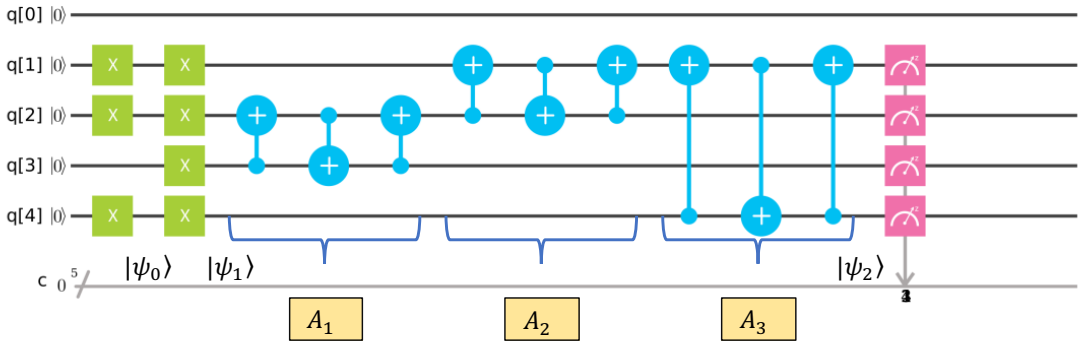

Figure 4.20. Experiment 4.7: 7x13 (mod 15). Adapted from [10].

[8]In contrast to most other situations, IBM uses q[4] to represent the least significant bit in this circuit. State $|\psi_0\rangle$ should then be read 13_{10} or 1101_2.

This circuit, however, cannot be run on the *ibmqx2*. IBM has constructed this gate using a fully connected *ideal simulator*, which is not constrained by the physical implementation of ***CNOT*** gates like the *ibmqx2*. There are three swap gates that used in this circuit, A_1, A_2, A_3. These swap gates use a series of ***CNOT*** gates to implement the swap. We will first examine the ***CNOT*** gates that are not supported on the *ibmqx2*; and then the swap gate itself. The second ***CNOT*** gate in the A_1 swap is not supported by the *ibmqx2*, refer to Figure 1.3. IBM provides a simple way to construct the necessary ***CNOT*** gate with the control and target qubits flipped. Figures 4.22 and 4.21 demonstrate this process. The circuit in Figure 4.21 was validated using *Mathematica* as shown in Appendix B.8.

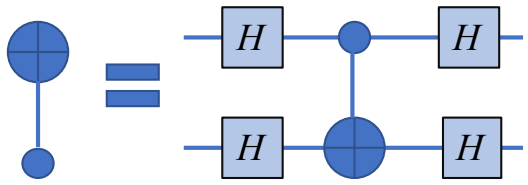

Figure 4.21. ***CNOT*** gate flip. Adapted from [10].

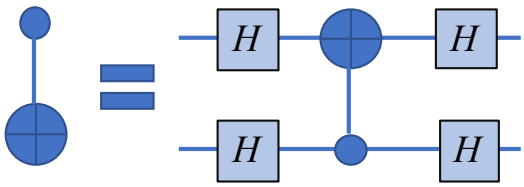

Figure 4.22. ***CNOT*** gate flip.

With the ability to alter the control and target qubit of the provided ***CNOT*** gates using the examples in Figures 4.21 and 4.22, we turn our attention to the swap gates. IBM provides a score to experiment with swap gates to see how they can be constructed even if a ***CNOT*** gate is not supported between the qubits that need to be swapped. This example is shown in Figure 4.23. This circuit swaps the state of q_0 in superposition with the q_1 in a state of $|0\rangle$. Although this is a contrived example, the necessary ***CNOT*** to swap directly between q_0 to q_1 is supported on the *ibmqx2*; it is a useful example of how to demonstrate swap gates and how they can be linked together.

[8]IBM provides example of a direct swap between q_0 and q_1 in the *Full User Guide*.

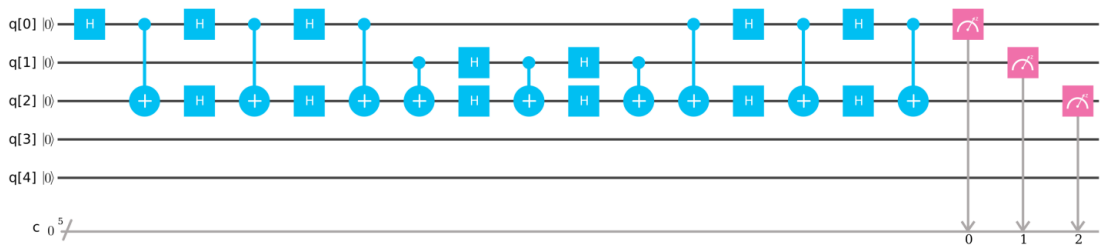

Figure 4.23. Swapping q_0 in superposition with q_1 in a $|0\rangle$ state. Source: [10].

One way to examine how the swaps work is to implement psuedo-code which steps through the logic. The code in Listing 4.1 implements the A_1 swap from Figure 4.26. The starting states of q_2 and q_3 are $|1\rangle$ and $|0\rangle$, respectively.

Listing 4.1: Swap gate psuedo-code

```
int main(){

    int q_2 = 1;
    int q_3 = 0;

    if (q_3 == 1){          //CNOT, ctrl = q_3, trgt = q_2
        if (q_2 == 1){
            q_2 = 0         //Flip qubit from 1 to 0
        }else{
            q_2 = 1         //Flip qubit from 0 to 1
        }
    }

    if (q_2 == 1){          //CNOT, ctrl = q_2, trgt = q_3
        if (q_3 == 1){
            q_3 = 0         //Flip qubit from 1 to 0
        }else{
            q_3 = 1         //Flip qubit from 0 to 1
```

```
        }
    }

     if (q_3 == 1){        //CNOT, ctrl = q_3, trgt = q_2
        if (q_2 == 1){
            q_2 = 0        //Flip qubit from 1 to 0
        }else{
            q_2 = 1        //Flip qubit from 0 to 1
            }
    }
    printf("q_2 = \%d, q_3 = \%d", q_2, q_3);
    return 0;
}
```

If the logic of this code were executed, the resulting print statement would output "`q_2 = 0, q_3 = 1`." The three alternate starting states can easily be verified by walking them through the example to understand how this process works. The score provided by IBM, shown in Figure 4.23, works as intended, but a more straightforward example can be constructed. By replacing the leading ***H*** gate applied to q_0 with an ***X*** gate, we can examine the circuit as it swaps a $|1\rangle$ state rather than swapping a state of superposition. This new example is shown in Figure 4.24.

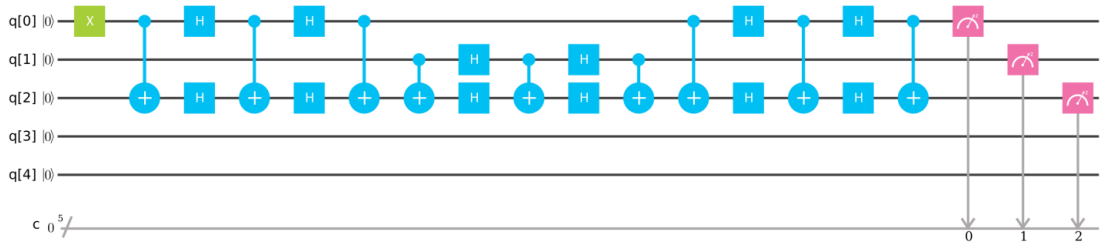

Figure 4.24. Experiment 4.8: Swapping q_0 in a $|1\rangle$ state and q_1 a $|0\rangle$ state.

As in the example provided in IBM, Figure 4.24 swaps the states of q_0 and q_1. This is accomplished in three separate swaps. The first swap is between q_0 and q_2, the next between

q_1 and q_2, and finally between q_0 and q_2. With q_0 being initialized to $|1\rangle$, this state will be transferred to q_1. The other qubit, q_2, is treated almost like a local variable that can be accessed by both q_0 and q_1. The results from running the score from Figure 4.24 on the *ideal simulator* are shown in Figure 4.25. These results match the testing conducting using *Mathematica* as shown in Appendix B.

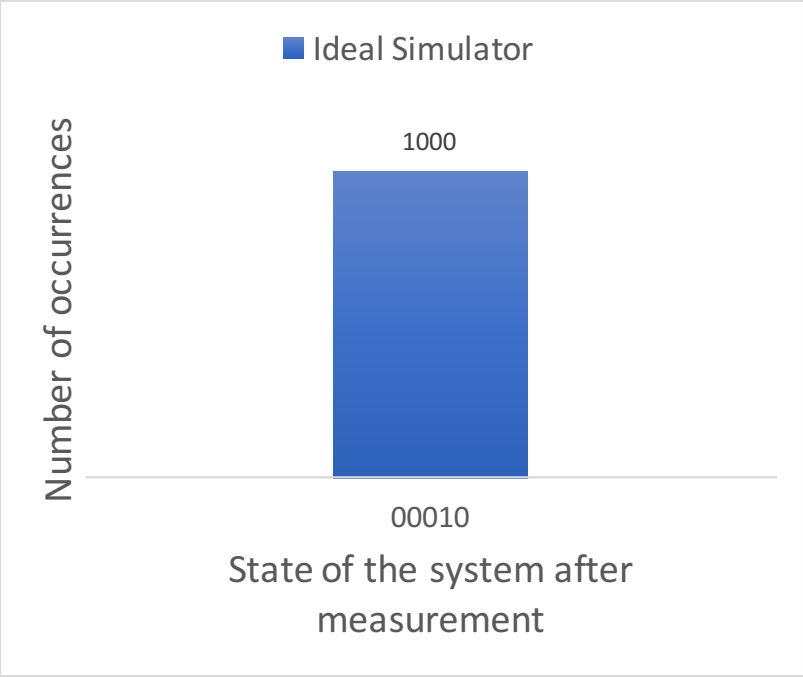

Figure 4.25. Results from Experiment 4.8.

With the ***CNOT*** and swap gates verified, it is now possible to build a circuit for modular multiplication that could be implemented directly on the *ibmqx2*. Such a circuit is shown in Figure 4.26. To modify the swap gates A_1 and A_2 of Figure 4.26, the second ***CNOT*** gate of each must substituted with ***CNOT*** gate shown in Figure 4.22. Swap gate A_3 is implemented by linking multiple swap gates together. Figure 1.3 shows that there is no ***CNOT*** between q_1 and q_4, so a swap between the two cannot be directly achieved. To circumvent this, we need to find a qubit for which both q_1 and q_4 have a ***CNOT*** gate implemented. The qubit, q_2, meets this requirement. The desired swap, A_3, is accomplished with three intermediate swaps, a_1, a_2, and a_3. This series of three swaps provides the desired effect of swapping q_2 and q_4. Figure 4.26 shows the full circuit capable of being run on the *ibmqx2*.

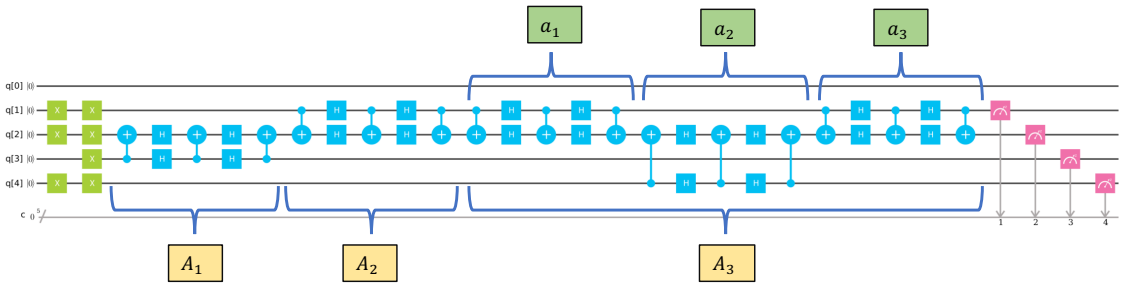

Figure 4.26. Experiment 4.9: 7x13 (mod 15) configured for the *ibmqx2*.

The two scores for modular multiplication, Experiment 4.7 and 4.9, were simulated using a fully connected simulator and on a simulator with the same connectivity as the *ibmqx2*, respectively. The results of these experiments are shown in Figure 4.27.

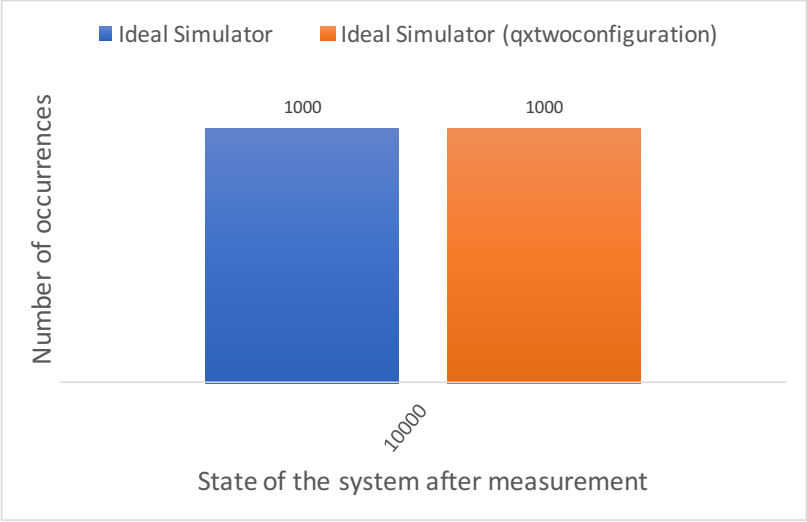

Figure 4.27. Results from Experiment 4.7 and 4.9: 7x13 (mod 15).

4.3.1 Quantum Fourier Transform

The final piece to be evaluated for Shor's algorithm is the QFT. This is the only score not found in the *Full User Guide* evaluated in this thesis. It is unclear why IBM choose to exclude the QFT from their user guide while providing a discussion of Shor's algorithm. The QASM for this code was found in [23] and is available in Appendix A. The purpose of applying the QFT is to return the period of the system [18]. Nielsen and Chuang provide a

general circuit for a QFT using n qubits. A specific example for a three-qubit QFT is shown in Figure 4.28. The QFT provided by IBM is shown in Figure 4.29.

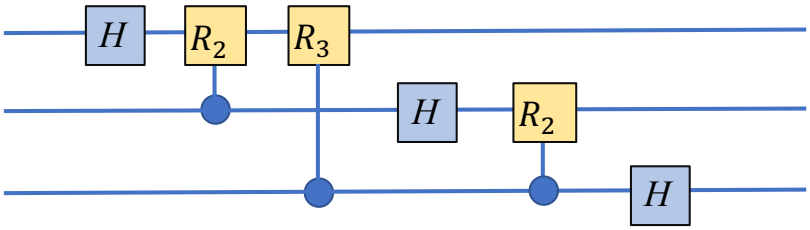

Figure 4.28. Three-qubit QFT.

$$R_k = \begin{bmatrix} 1 & 0 \\ 0 & e^{2\pi i/2^k} \end{bmatrix} \tag{4.9}$$

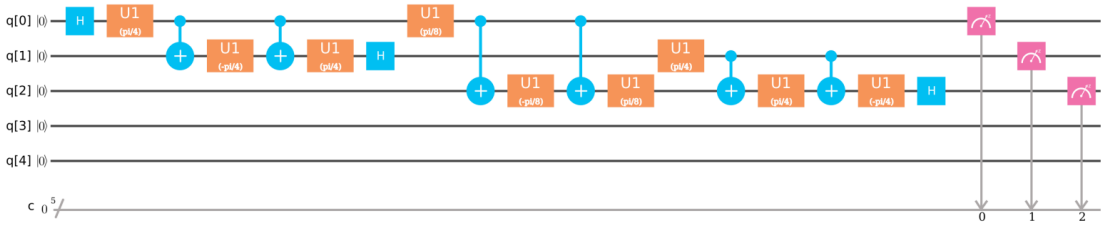

Figure 4.29. Experiment 4.10: Quantum Fourier Transform. Source: [23].

With Figure 4.28 and Equation 4.9, we can begin to construct the the necessary matrix to represent the controlled-$\boldsymbol{R}$ gates. The R_2 and R_3 gates are defined in Equations 4.10 and 4.11, respectively. Euler's formula, $e^{i\theta} = \cos\theta + i\sin\theta$, is used to simplify the $\boldsymbol{R}_2$ gate.

$$\begin{aligned} R_2 &= \begin{bmatrix} 1 & o \\ 0 & e^{2\pi i/2^2} \end{bmatrix} \\ &= \begin{bmatrix} 1 & o \\ 0 & i \end{bmatrix} \end{aligned} \tag{4.10}$$

$$
\begin{aligned}
R_3 &= \begin{bmatrix} 1 & o \\ 0 & e^{2\pi i/2^3} \end{bmatrix} \\
&= \begin{bmatrix} 1 & o \\ 0 & e^{\pi i/4} \end{bmatrix}
\end{aligned}
\tag{4.11}
$$

These two ***R*** gates need to then be applied to the target qubit when the control-qubit is $|1\rangle$. The matrices for these are defined in Equations 4.12 and 4.13, but those matrices must be defined in terms of the gates supplied by IBM. They cannot be implemented directly on *ibmqx2* to compare the QFT provided by Nielsen and Chuang with that from IBM.

$$
\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & i & 0 \end{bmatrix}
\tag{4.12}
$$

$$
\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & e^{\frac{i\pi}{4}} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & e^{\frac{i\pi}{4}} \end{bmatrix}
\tag{4.13}
$$

The scores from Figure 4.28 and 4.29 do not immediately look similar, but, as will be shown, actually represent the same circuit. The main difference between these two scores is the way in which the controlled-***R*** gates are implemented. The only four-by-four matrix that IBM has implemented on the *ibmqx2* is the ***CNOT*** gate. The controlled-***R*** gates must be therefore be composed of two-by-two matrices and the available ***CNOT***. Lemma 5.1 of [22]

states that "for a unitary 2x2 matrix W, a $\wedge_1 W$ gate[9] can be simulated by a network of the form

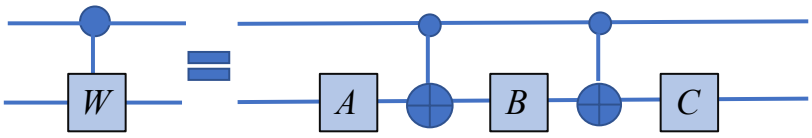

Figure 4.30. From lemma 5.1. Adapted from [22].

where A, B, and $C \epsilon$ SU(2) if and only if W ϵ SU(2)." The SU(2) matrices referred to are a special unitary group of 2-by-2 matrices [39]. Further A, B, and C must be defined such that $ABC = i$ [10]. With this formula, it is possible to construct any necessary control gate from the basis provided. Note that the $\boldsymbol{R}_2$ gate is equal to an $\boldsymbol{S}$ gate, as shown in Equation 2.9, and the $\boldsymbol{R}_3$ gate is equal to a $\boldsymbol{T}$ gate, as shown in Equation 2.10. A controlled-$\boldsymbol{S}$ gate can be constructed with Lemma 5.1 within the basis provided by IBM. Figure 4.31 shows how a controlled-$\boldsymbol{S}$ gate or $\boldsymbol{R}_2$ gate can be constructed. This was verified using *Mathematica* and is shown in Appendix B.

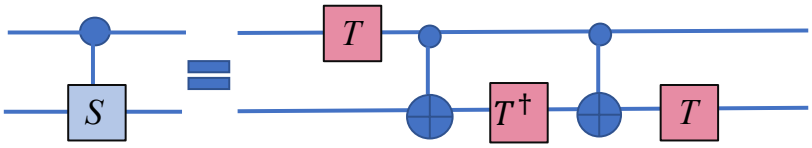

Figure 4.31. Constructing a controlled-$\boldsymbol{S}$ from the gates provided by IBM.

This can be similarly done for a control-$\boldsymbol{T}$ gate. With this method the circuit for the IBM QFT can be rewritten in a simplified form as shown in Figure 4.32. This should now follow the form exactly from Nielsen and Chuang, and when examined using *Mathematica*, the resulting matrices of the two QFTs are found to be the same.

[9]The notation $\wedge_1$ is used to identify a control gate with one control qubit. A ***CNOT*** gate in this form would be written as $\wedge_1\boldsymbol{X}$, and the Toffoli gate as $\wedge_2\boldsymbol{X}$.

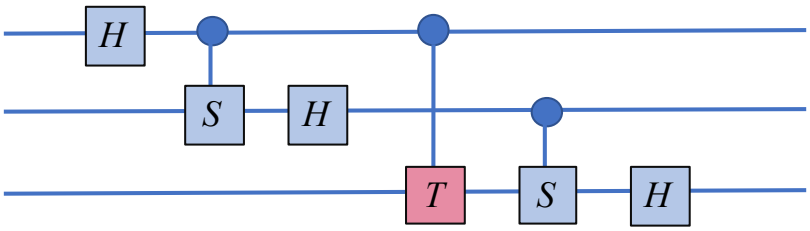

Figure 4.32. IBM QFT simplified.

The results from comparing the score from Figure 4.28 and Experiment 4.10 are shown in Figure 4.33 and 4.34, respectively. The full notebooks for these circuits are found in Appendix B.

In[13]:= **T6.T5.T4.T3.T2.T1**

Out[13]=
$$\left\{\left\{\frac{1}{2\sqrt{2}}, \frac{1}{2\sqrt{2}}, \frac{1}{2\sqrt{2}}, \frac{1}{2\sqrt{2}}, \frac{1}{2\sqrt{2}}, \frac{1}{2\sqrt{2}}, \frac{1}{2\sqrt{2}}, \frac{1}{2\sqrt{2}}\right\},\right.$$
$$\left\{\frac{1}{2\sqrt{2}}, -\frac{1}{2\sqrt{2}}, \frac{1}{2\sqrt{2}}, -\frac{1}{2\sqrt{2}}, \frac{1}{2\sqrt{2}}, -\frac{1}{2\sqrt{2}}, \frac{1}{2\sqrt{2}}, -\frac{1}{2\sqrt{2}}\right\},$$
$$\left\{\frac{1}{2\sqrt{2}}, \frac{i}{2\sqrt{2}}, -\frac{1}{2\sqrt{2}}, -\frac{i}{2\sqrt{2}}, \frac{1}{2\sqrt{2}}, \frac{i}{2\sqrt{2}}, -\frac{1}{2\sqrt{2}}, -\frac{i}{2\sqrt{2}}\right\},$$
$$\left\{\frac{1}{2\sqrt{2}}, -\frac{i}{2\sqrt{2}}, -\frac{1}{2\sqrt{2}}, \frac{i}{2\sqrt{2}}, \frac{1}{2\sqrt{2}}, -\frac{i}{2\sqrt{2}}, -\frac{1}{2\sqrt{2}}, \frac{i}{2\sqrt{2}}\right\},$$
$$\left\{\frac{1}{2\sqrt{2}}, \frac{e^{\frac{i\,PI}{4}}}{2\sqrt{2}}, \frac{i}{2\sqrt{2}}, \frac{i\,e^{\frac{i\,PI}{4}}}{2\sqrt{2}}, -\frac{1}{2\sqrt{2}}, -\frac{e^{\frac{i\,PI}{4}}}{2\sqrt{2}}, -\frac{i}{2\sqrt{2}}, -\frac{i\,e^{\frac{i\,PI}{4}}}{2\sqrt{2}}\right\},$$
$$\left\{\frac{1}{2\sqrt{2}}, -\frac{e^{\frac{i\,PI}{4}}}{2\sqrt{2}}, \frac{i}{2\sqrt{2}}, -\frac{i\,e^{\frac{i\,PI}{4}}}{2\sqrt{2}}, -\frac{1}{2\sqrt{2}}, \frac{e^{\frac{i\,PI}{4}}}{2\sqrt{2}}, -\frac{i}{2\sqrt{2}}, \frac{i\,e^{\frac{i\,PI}{4}}}{2\sqrt{2}}\right\},$$
$$\left\{\frac{1}{2\sqrt{2}}, \frac{i\,e^{\frac{i\,PI}{4}}}{2\sqrt{2}}, -\frac{i}{2\sqrt{2}}, \frac{e^{\frac{i\,PI}{4}}}{2\sqrt{2}}, -\frac{1}{2\sqrt{2}}, -\frac{i\,e^{\frac{i\,PI}{4}}}{2\sqrt{2}}, \frac{i}{2\sqrt{2}}, -\frac{e^{\frac{i\,PI}{4}}}{2\sqrt{2}}\right\},$$
$$\left.\left\{\frac{1}{2\sqrt{2}}, -\frac{i\,e^{\frac{i\,PI}{4}}}{2\sqrt{2}}, -\frac{i}{2\sqrt{2}}, -\frac{e^{\frac{i\,PI}{4}}}{2\sqrt{2}}, -\frac{1}{2\sqrt{2}}, \frac{i\,e^{\frac{i\,PI}{4}}}{2\sqrt{2}}, \frac{i}{2\sqrt{2}}, \frac{e^{\frac{i\,PI}{4}}}{2\sqrt{2}}\right\}\right\}$$

Figure 4.33. *Mathematica* results for QFT, as presented by Nielsen and Chuang.

In[27]:= **T18.T17.T16.T15.T14.T13.T12.T11.T10.T9.T8.T7.T6.T5.T4.T3.T2.T1**

Out[27]=
$$\left\{\left\{\frac{1}{2\sqrt{2}}, \frac{1}{2\sqrt{2}}, \frac{1}{2\sqrt{2}}, \frac{1}{2\sqrt{2}}, \frac{1}{2\sqrt{2}}, \frac{1}{2\sqrt{2}}, \frac{1}{2\sqrt{2}}, \frac{1}{2\sqrt{2}}\right\},\right.$$
$$\left\{\frac{1}{2\sqrt{2}}, -\frac{1}{2\sqrt{2}}, \frac{1}{2\sqrt{2}}, -\frac{1}{2\sqrt{2}}, \frac{1}{2\sqrt{2}}, -\frac{1}{2\sqrt{2}}, \frac{1}{2\sqrt{2}}, -\frac{1}{2\sqrt{2}}\right\},$$
$$\left\{\frac{1}{2\sqrt{2}}, \frac{e^{\frac{i\,PI}{2}}}{2\sqrt{2}}, -\frac{1}{2\sqrt{2}}, -\frac{e^{\frac{i\,PI}{2}}}{2\sqrt{2}}, \frac{1}{2\sqrt{2}}, \frac{e^{\frac{i\,PI}{2}}}{2\sqrt{2}}, -\frac{1}{2\sqrt{2}}, -\frac{e^{\frac{i\,PI}{2}}}{2\sqrt{2}}\right\},$$
$$\left\{\frac{1}{2\sqrt{2}}, -\frac{e^{\frac{i\,PI}{2}}}{2\sqrt{2}}, -\frac{1}{2\sqrt{2}}, \frac{e^{\frac{i\,PI}{2}}}{2\sqrt{2}}, \frac{1}{2\sqrt{2}}, -\frac{e^{\frac{i\,PI}{2}}}{2\sqrt{2}}, -\frac{1}{2\sqrt{2}}, \frac{e^{\frac{i\,PI}{2}}}{2\sqrt{2}}\right\},$$
$$\left\{\frac{1}{2\sqrt{2}}, \frac{e^{\frac{i\,PI}{4}}}{2\sqrt{2}}, \frac{e^{\frac{i\,PI}{2}}}{2\sqrt{2}}, \frac{e^{\frac{3\,i\,PI}{4}}}{2\sqrt{2}}, -\frac{1}{2\sqrt{2}}, -\frac{e^{\frac{i\,PI}{4}}}{2\sqrt{2}}, -\frac{e^{\frac{i\,PI}{2}}}{2\sqrt{2}}, -\frac{e^{\frac{3\,i\,PI}{4}}}{2\sqrt{2}}\right\},$$
$$\left\{\frac{1}{2\sqrt{2}}, -\frac{e^{\frac{i\,PI}{4}}}{2\sqrt{2}}, \frac{e^{\frac{i\,PI}{2}}}{2\sqrt{2}}, -\frac{e^{\frac{3\,i\,PI}{4}}}{2\sqrt{2}}, -\frac{1}{2\sqrt{2}}, \frac{e^{\frac{i\,PI}{4}}}{2\sqrt{2}}, -\frac{e^{\frac{i\,PI}{2}}}{2\sqrt{2}}, \frac{e^{\frac{3\,i\,PI}{4}}}{2\sqrt{2}}\right\},$$
$$\left\{\frac{1}{2\sqrt{2}}, \frac{e^{\frac{3\,i\,PI}{4}}}{2\sqrt{2}}, -\frac{e^{\frac{i\,PI}{2}}}{2\sqrt{2}}, -\frac{e^{\frac{5\,i\,PI}{4}}}{2\sqrt{2}}, -\frac{1}{2\sqrt{2}}, -\frac{e^{\frac{3\,i\,PI}{4}}}{2\sqrt{2}}, \frac{e^{\frac{i\,PI}{2}}}{2\sqrt{2}}, \frac{e^{\frac{5\,i\,PI}{4}}}{2\sqrt{2}}\right\},$$
$$\left.\left\{\frac{1}{2\sqrt{2}}, -\frac{e^{\frac{3\,i\,PI}{4}}}{2\sqrt{2}}, -\frac{e^{\frac{i\,PI}{2}}}{2\sqrt{2}}, \frac{e^{\frac{5\,i\,PI}{4}}}{2\sqrt{2}}, -\frac{1}{2\sqrt{2}}, \frac{e^{\frac{3\,i\,PI}{4}}}{2\sqrt{2}}, \frac{e^{\frac{i\,PI}{2}}}{2\sqrt{2}}, -\frac{e^{\frac{5\,i\,PI}{4}}}{2\sqrt{2}}\right\}\right\}$$

Figure 4.34. *Mathematica* results for QFT, as presented by IBM.

CHAPTER 5:
Conclusion

This thesis has conducted a variety of experiments using the architecture of the *ibmqx2*. Experiments 3.3 - 3.6 were able to demonstrate that the *ibmqx2* creates a reliable state of superposition. The results from Experiment 3.6 demonstrated an unexpected periodicity. There could be several explanations for the unexpected results when the score from Figure 3.19 was run with 4096 and 8192 shots. This could have been a result of the error rates compounding over multiple experiments and were less noticeable with a lower shot count.

The second major quantum principle demonstrated was entanglement. Experiment 3.7 demonstrated this through creating the *Bell* state $|\beta_{00}\rangle$. These two capabilities form the building blocks of many quantum algorithms. This thesis does not address in sufficient detail the quality of the qubits implemented by IBM, but the initial results are promising.

The limited number of gates implemented by IBM is sufficient to build any additional gates (i.e., control-***Z*** gates) with the basis provided, within reason. For example, it would be unreasonable to expect the *ibmqx2* to support a six-by-six unitary matrix. The approach by IBM to construct a limited number of physical quantum gates that can be manipulated to construct any arbitrary gate seems a logical starting point. The published information evaluated in writing this thesis did not uncover a reason why IBM chose to abstract certain quantum gates instead of creating dedicated gates. Their approach would likely scale well with additional qubits. It would potentially reduce the number of gates that would have to be constructed and the possible vectors for failures.

Experimenting with the *ibmqx2* presented some interesting challenges. One of the most interesting was the decomposition of a unitary matrix into elementary quantum gates. This was the motivation for the use of *Mathematica* in this thesis. While texts like *Quantum Computing for Computer Scientists* are helpful for understanding the theory, they frequently use compiled gates. These cannot be directly implemented, as in *Mathematica*, but must be constructed using the basis as implemented by the architecture. The use of tools such as *Qubiter* to accomplish this is further discussed in Chapter 7. Similarly, the ability to construct ***CNOT*** gates in the reversed direction and swap gates provides a great deal of

flexibility. The level of control the user has on this experimental system is comparable to that achieved when programming in assembly language. There are some constructs, like loops, that are not directly available in a single assembly instruction, but can be supported through the correct use of the available instructions.

The results returned from IBM do conform with the expected error rates on a physical quantum computer. However, there is no way to confirm that these results are from the *ibmqx2* and not a simulator that has artificial noise injected to provide more realistic results. This question has been asked on the *IBM Quantum Experience* community forum, but has not yet been responded to. For the purpose of this thesis, this is not necessary. Testing how to program circuits and validating that those circuits perform as expected does not need to be conducted on a real processor. For future work and investing in quantum technology, the DoD will need to validate that these results are being measured from a real quantum processor. There are several ways in which this could be reasonably accomplished. One such method would be to sign the results from the quantum computer before they are exported. Additionally, after reliable quantum computers break the 50-qubit threshold, they will begin to exceed the processing power supercomputers are able to simulate. In 2007, testing was done using parallel computers to simulate "up to a 36 qubits, using up to 4096 processors and up to 1 TB of memory" [40]. IBM announced in 2017 that they were able to simulate a 56-qubit quantum computer using the Vulcan supercomputer [41]. The ability to simulate quantum results of smaller quantum computers should make it possible for false results to be returned. This is a short-term problem, though. Increasing the number of qubits by one doubles the potential number of computations that is necessary, and even the most advanced supercomputers will have difficulties simulating those algorithms. At that point users could be reasonably assured that their results are not being returned from a simulation.

The error rates, particularly in Experiments 3.5a and 4.4, show that there is still room for improvement with the *ibmqx2*. There are techniques for error correction that IBM provides in the *Full User Guide*, to compensate for some of these errors. This would increase the overhead of running any particular circuit. Even with this limitation, there are still many positive aspects of the design of the *IBM Quantum Experience*. The *Beginner's Guide* and *Full User Guide* provide an excellent reference for understanding the basics of quantum computing. The explanations and examples provided closely follow the quantum circuit

model, as presented by Yanofsky in [18]. With the exception of the ***IF***, necessary to demonstrate teleportation, the *ibmqx2* was able to adequately execute all tested circuits. After many months of research and experimentation, we feel confident that the IBM *ibmqx2* is exploiting quantum mechanics during runtime. Further, the IBM *ibmqx2* and more broadly the *IBM Quantum Experience* provide a user-friendly environment which is helpful in entering the field of quantum computing. While the Naval Postgraduate School is working to establish a formal quantum information science program, the tools and information provided by IBM provide an excellent open-source option for computer science students wishing to explore quantum computing.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 6:
Related Work

The field of quantum computing is at an exciting stage. Companies like IBM are investing in quantum technologies and disrupting the field with unique approaches. One such company, which has made a significant impact on the quantum industry, is D-Wave.

D-Wave has been involved in the field of quantum computing for nearly 20 years. They have released several commercial quantum computers. Their most recent quantum computer was announced in late 2017, the D-Wave 2000Q system [42]. The D-Wave quantum computing systems are based on the quantum annealing computational model, which contrasts with the quantum circuit computation model presented in this thesis. In one of their trade publications, D-Wave describes quantum annealing as follows: "it harnesses the natural tendency of real-world quantum systems to find low-energy states" [43]. The computer is initialized into a starting state and then is annealed "toward the problem to be solved such that it remains in a low energy state throughout the process" [43]. The final resting state of the system would then be interpreted as the solution being sought. D-Wave has attracted much attention for its progress in quantum computing.

Not all of that attention has been positive. Aaronson points out several of the critiques of the claims from D-Wave, including their methods for calculating quantum speed-up and the algorithms they compared their quantum computer against [44]. There has been much debate about how to evaluate the claims of quantum speed-up that D-Wave has announced over the years [45], [46]. Even with the controversies over the "quantum-ness" and performance of the D-Wave systems, they are exploring technologies which could be used to help advance the quantum industry. These kinds of advances are what Preskill might call Noisy Intermediate-Scale Quantum (NISQ) technologies [47]. His coining of the term comes as word of caution and of optimism. Technologies like the D-Wave 2000Q or the IBM *ibmqx2* have issues that will need to be addressed; however, they provide opportunities as well. Preskill emphasizes the need for "progress in the near term by developing better methods and hardware for implementing quantum error correction" as a means to provide improved utility of current quantum systems [47].

Excluding D-Wave, most other companies developing quantum computers have fewer than 50 qubits. One of the numerous challenges with creating larger quantum computers is maintaining coherence across those qubits. One way to address scalability challenges is through distributed computing. With the development of quantum communication channels, it is possible to create a distributed quantum computing network. Researchers have been able to create entangled states between atoms and photons [48]. There are challenges with working with distributed computers, but until large fault-tolerant quantum computers are readily available, they can provide a valuable intermediate solution for quantum computing.

This thesis is one of several works that has attempted to baseline the quantum computers released by IBM. Michielsen et al. uses four separate classes, entanglement, two-qubit + two-qubit adder, identity operations, and error correction, to test the *IBM Quantum Experience* [49]. They encountered errors with their results, some of which they were able to attribute to a specific calibration, and others were unexplained. They also note the importance of benchmarking additional quantum computers, especially against those which have chosen a different qubit implementation. Devitt similarly conducts a series of experiments on the *IBM Quantum Experience* [50]. He uses "quantum error correction, quantum arithmetic, quantum graph theory, and fault-tolerant quantum computation....While the results are subject to significant noise, the correct results are returned" [50]. These results do concur with those observed in Experiments 3.5a and 4.4. Both agree that even with the current limitations and issues with error rates, the *IBM Quantum Experience* provides a valuable experience and is a necessary data point for future benchmarking.

CHAPTER 7:
Future Work

7.1 QISKit

Beyond the *Composer* and QASM editor, IBM has provided a Python toolkit to program the quantum processors. This toolkit, called *QISKit*, is the only current means to access the 16-qubit processor. With a familiarization of the *QISKit* and access to the 16-qubit, it would be possible to construct an implementation of Shor's algorithm. An interesting experiment would be a comparison of the different implementations of Shor's algorithm.

7.2 Qubiter

There are multiple challenges with implementing quantum algorithms. One such challenge is developing the necessary unitary matrices from elementary quantum gates. Defining the matrix is not a particularly difficult task; however, it does become tedious when dealing with large matrices. This process could easily be accomplished with automation. In researching this thesis, a product was uncovered that assists with the decomposition. *Qubiter* is able to decompose a given unitary matrix into elementary quantum gates compatible with a defined connectivity. This could be a valuable tool for experimentation with algorithms and finding more efficient means of constructing the necessary unitary matrices used.

7.3 High-Level Quantum Computing Language

The QASM language used by IBM is comparable to other low-level assembly languages. It allows near complete control of how the circuit executes. This puts the burden of efficiency on the programmer. Many high-level languages abstract away low-level processes and permit the programmer to focus on the program at a higher level. Quantum computing could benefit from additional research in this field. Just as 64-bit x86 code does not run on a 32-bit x86 processor, the QASM code in Appendix A cannot be transferred directly to *ibmqx4* or other chips unless the exact ***CNOT*** gates from the *ibmqx2* are supported. This lack of backward compatibility could be overcome by developing a higher-level coding

language. As discussed in Section 4.3, there are efficient ways to create swap gates and ***CNOT*** gates. These types of techniques could be implemented in a high-level language, freeing the programmer from the need to consider the physical architecture of the processor until compile time. There are established quantum programming languages [51]. An example of how to program a quantum computer using Quantum Programming Language (QPL) is demonstrated in [52]. Developing a set of libraries that adopt the existing functions of QPL to a specific quantum processor's configuration could allow the code to be easily portable. When switching from one quantum processor to another, the programmer would only need to adjust the libraries that are called to conform to the new processor.

APPENDIX A:
QASM Code

A.1 Experiment 3.1

```
//OPENQASM 2.0
IBMQASM 2.0;

include "qelib1.inc";

qreg q[5];
creg c[5];

measure q[0] -> c[0];
measure q[1] -> c[1];
measure q[2] -> c[2];
measure q[3] -> c[3];
measure q[4] -> c[4];
```

A.2 Experiment 3.2

```
//OPENQASM 2.0
IBMQASM 2.0;

include "qelib1.inc";

qreg q[5];
creg c[5];

x q[0];
x q[1];
x q[2];
x q[3];
```

```
x q[4];
measure q[0] -> c[0];
measure q[1] -> c[1];
measure q[2] -> c[2];
measure q[3] -> c[3];
measure q[4] -> c[4];
```

A.3 Experiment 3.3

```
//OPENQASM 2.0
IBMQASM 2.0;

include "qelib1.inc";

qreg q[5];
creg c[5];

h q[0];
measure q[0] -> c[0];
```

A.4 Experiment 3.4

```
//OPENQASM 2.0
IBMQASM 2.0;

include "qelib1.inc";

qreg q[5];
creg c[5];

h q[0];
measure q[0] -> c[0];
measure q[1] -> c[1];
measure q[2] -> c[2];
measure q[3] -> c[3];
```

```
measure q[4] -> c[4];
```

A.5 Experiment 3.5

```
//OPENQASM 2.0
IBMQASM 2.0;

include "qelib1.inc";

qreg q[5];
creg c[5];

h q[0];
h q[1];
h q[2];
h q[3];
h q[4];
measure q[0] -> c[0];
measure q[1] -> c[1];
measure q[2] -> c[2];
measure q[3] -> c[3];
measure q[4] -> c[4];
```

A.5.1 Experiment 3.5a

This experiment uses the same QASM code as in Experiment 3.5. The shot parameters for this experiment are adjusted through the settings options for the *'run'*, *ibmqx2*, or *'simulate'*, *ideal simulator*.

A.6 Experiment 3.7

```
//OPENQASM 2.0
IBMQASM 2.0;

include "qelib1.inc";
```

```
qreg q[5];
creg c[5];

h q[0];
cx q[0],q[1];
measure q[0] -> c[0];
measure q[1] -> c[1];
measure q[2] -> c[2];
measure q[3] -> c[3];
measure q[4] -> c[4];
```

A.7 Experiment 4.1

```
//OPENQASM 2.0
IBMQASM 2.0;

include "qelib1.inc";

qreg q[5];
creg c[5];

h q[0];
h q[1];
h q[2];
h q[0];
h q[1];
h q[2];
measure q[0] -> c[0];
measure q[1] -> c[1];
measure q[2] -> c[2];
```

A.8 Experiment 4.2

```
//OPENQASM 2.0
IBMQASM 2.0;

include "qelib1.inc";

qreg q[5];
creg c[5];

h q[0];
h q[1];
h q[2];
h q[2];
z q[0];
cx q[1],q[2];
h q[2];
h q[0];
h q[1];
h q[2];
measure q[0] -> c[0];
measure q[1] -> c[1];
measure q[2] -> c[2];
```

A.9 Experiment 4.3

```
//OPENQASM 2.0
IBMQASM 2.0;

include "qelib1.inc";

qreg q[5];
creg c[5];

h q[1];
h q[2];
s q[1];
```

```
s q[2];
h q[2];
cx q[1],q[2];
h q[2];
s q[1];
s q[2];
h q[1];
h q[2];
x q[1];
x q[2];
h q[2];
cx q[1],q[2];
h q[2];
x q[1];
x q[2];
h q[1];
h q[2];
measure q[1] -> c[1];
measure q[2] -> c[2];
```

A.10 Experiment 4.4

```
//OPENQASM 2.0
IBMQASM 2.0;

include "qelib1.inc";

qreg q[5];
creg c[5];

h q[1];
h q[2];
s q[2];
h q[2];
cx q[1],q[2];
```

```
h q[2];
s q[2];
h q[1];
h q[2];
x q[1];
x q[2];
h q[2];
cx q[1],q[2];
h q[2];
x q[1];
x q[2];
h q[1];
h q[2];
measure q[1] -> c[1];
measure q[2] -> c[2];
```

A.11 Experiment 4.5

```
//OPENQASM 2.0
IBMQASM 2.0;

include "qelib1.inc";

qreg q[5];
creg c[5];

h q[1];
h q[2];
s q[1];
h q[2];
cx q[1],q[2];
h q[2];
s q[1];
h q[1];
h q[2];
```

```
x q[1];
x q[2];
h q[2];
cx q[1],q[2];
h q[2];
x q[1];
x q[2];
h q[1];
h q[2];
measure q[1] -> c[1];
measure q[2] -> c[2];
```

A.12 Experiment 4.6

```
//OPENQASM 2.0
IBMQASM 2.0;

include "qelib1.inc";

qreg q[5];
creg c[5];

h q[1];
h q[2];
h q[2];
cx q[1],q[2];
h q[2];
h q[1];
h q[2];
x q[1];
x q[2];
h q[2];
cx q[1],q[2];
h q[2];
x q[1];
```

```
x q[2];
h q[1];
h q[2];
measure q[1] -> c[1];
measure q[2] -> c[2];
```

A.13 Experiment 4.7

```
//OPENQASM 2.0
IBMQASM 2.0;

include "qelib1.inc";

qreg q[5];
creg c[5];

x q[1];
x q[2];
x q[4];
x q[1];
x q[2];
x q[3];
x q[4];
cx q[3],q[2];
cx q[2],q[3];
cx q[3],q[2];
cx q[2],q[1];
cx q[1],q[2];
cx q[2],q[1];
cx q[4],q[1];
cx q[1],q[4];
cx q[4],q[1];
measure q[1] -> c[1];
measure q[2] -> c[2];
measure q[3] -> c[3];
```

```
measure q[4] -> c[4];
```

*This code cannot be executed on the *ibmqx2*, it is only compatible with the *ideal simulator*.

A.14 Experiment 4.8

```
//OPENQASM 2.0
IBMQASM 2.0;

include "qelib1.inc";

qreg q[5];
creg c[5];

x q[0];
cx q[0],q[2];
h q[0];
h q[2];
cx q[0],q[2];
h q[0];
h q[2];
cx q[0],q[2];
cx q[1],q[2];
h q[1];
h q[2];
cx q[1],q[2];
h q[1];
h q[2];
cx q[1],q[2];
cx q[0],q[2];
h q[0];
h q[2];
cx q[0],q[2];
h q[0];
h q[2];
```

```
cx q[0],q[2];
measure q[0] -> c[0];
measure q[1] -> c[1];
measure q[2] -> c[2];
```

A.15 Experiment 4.9

```
//OPENQASM 2.0
IBMQASM 2.0;

include "qelib1.inc";

qreg q[5];
creg c[5];

x q[1];
x q[2];
x q[4];
x q[1];
x q[2];
x q[3];
x q[4];
cx q[3],q[2];
h q[2];
h q[3];
cx q[3],q[2];
h q[2];
h q[3];
cx q[3],q[2];
cx q[1],q[2];
h q[1];
h q[2];
cx q[1],q[2];
h q[1];
h q[2];
```

```
cx q[1],q[2];
cx q[1],q[2];
h q[1];
h q[2];
cx q[1],q[2];
h q[1];
h q[2];
cx q[1],q[2];
cx q[4],q[2];
h q[2];
h q[4];
cx q[4],q[2];
h q[2];
h q[4];
cx q[4],q[2];
cx q[1],q[2];
h q[1];
h q[2];
cx q[1],q[2];
h q[1];
h q[2];
cx q[1],q[2];
measure q[1] -> c[1];
measure q[2] -> c[2];
measure q[3] -> c[3];
measure q[4] -> c[4];
```

A.16 Experiment 4.10

```
//OPENQASM 2.0
IBMQASM 2.0;

include "qelib1.inc";

qreg q[5];
```

```
creg c[5];

h q[0];

// cu1(pi/2) q[0],q[1];
u1(pi/4) q[0];
cx q[0],q[1];
u1(-pi/4) q[1];
cx q[0],q[1];
u1(pi/4) q[1];
// end cu1

h q[1];

// cu1(pi/4) q[0],q[2];
u1(pi/8) q[0];
cx q[0],q[2];
u1(-pi/8) q[2];
cx q[0],q[2];
u1(pi/8) q[2];
// end cu1

// cu1(pi/2) q[1],q[2];
u1(pi/4) q[1];
cx q[1],q[2];
u1(pi/4) q[2];
cx q[1],q[2];
u1(-pi/4) q[2];
// end cu1

h q[2];
measure q[0] -> c[0];
measure q[1] -> c[1];
measure q[2] -> c[2];
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX B:
Mathematica Notebooks

B.1 Experiment 4.1 *Mathematica* Notebook.

```
H = 1/(2^(1/2))*{{1,1}, {1,-1}}
ID = {{1,0}, {0,1}}
CNOT = {{1,0,0,0}, {0,1,0,0}, {0,0,0,1}, {0,0,1,0}}
Z = {{1,0}, {0,-1}
A0 = {1,0,0,0,0,0,0,0}
T1 = KroneckerProduct[ID,ID,H]
T2 = KroneckerProduct[Z,CNOT]
T0 = KroneckerProduct[H,H,H]

T0.T0.A0


Out = {1,0,0,0,0,0,0,0}
```

B.2 Computing U_f in Experiment 4.2 *Mathematica* Notebook.

```
H = 1/(2^(1/2))*{{1,1}, {1,-1}}
ID = {{1,0}, {0,1}}
CNOT = {{1,0,0,0}, {0,1,0,0}, {0,0,0,1}, {0,0,1,0}}
Z = {{1,0}, {0,-1}

T1 = KroneckerProduct[ID,ID,H]
T2 = KroneckerProduct[Z,CNOT]
T1.T2.T1

Out= {{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, -1, 0},
{0, 0, 0, 0, 0, 0, 0, 1}}
```

B.3 Experiment 4.2 *Mathematica* Notebook.

```
H = 1/(2^(1/2))*{{1,1}, {1,-1}}
ID = {{1,0}, {0,1}}
CNOT = {{1,0,0,0}, {0,1,0,0}, {0,0,0,1}, {0,0,1,0}}
Z = {{1,0}, {0,-1}
A0 = {1,0,0,0,0,0,0,0}
T1 = KroneckerProduct[ID,ID,H]
T2 = KroneckerProduct[Z,CNOT]
T0 = KroneckerProduct[H,H,H]

T0.T1.T2.T1.T0.A0
```

$$\text{Out} = \left\{0,0,0,0,\tfrac{1}{2},\tfrac{1}{2},\tfrac{1}{2},-\tfrac{1}{2}\right\}$$

B.4 Experiment 4.3 *Mathematica* Notebook.

```
H = 1/(2^(1/2))*{{1,1}, {1,-1}}
ID = {{1,0}, {0,1}}
S = {{1,0}, {0,I}}
X = {{0,1}, {1,0}}
CNOT = {{1,0,0,0}, {0,1,0,0}, {0,0,0,1}, {0,0,1,0}}
HH = KroneckerProduct[H,H]
SS = KronekcerProduct[S,S]
IH = KroneckerProduct[ID,H]
CT = CNOT
XX = KroneckerProduct[X,X]
Phi0 = {1,0,0,0}
```

```
HH.XX.IH.CT.IH.XX.HH.SS.IH.CT.IH.SS.HH.Phi0


Out = {1,0,0,0}
```

B.5 Experiment 4.4 *Mathematica* Notebook.

```
H = 1/(2^(1/2))*{{1,1}, {1,-1}}
ID = {{1,0}, {0,1}}
S = {{1,0}, {0,I}}
X = {{0,1}, {1,0}}
CNOT = {{1,0,0,0}, {0,1,0,0}, {0,0,0,1}, {0,0,1,0}}
HH = KroneckerProduct[H,H]
IS = KronekcerProduct[I,S]
IH = KroneckerProduct[ID,H]
CT = CNOT
XX = KroneckerProduct[X,X]
Phi0 = {1,0,0,0}

HH.XX.IH.CT.IH.XX.HH.IS.IH.CT.IH.IS.HH.Phi0


Out = {0,-1,0,0}
```

B.6 Experiment 4.5 *Mathematica* Notebook.

```
H = 1/(2^(1/2))*{{1,1}, {1,-1}}
ID = {{1,0}, {0,1}}
S = {{1,0}, {0,I}}
X = {{0,1}, {1,0}}
CNOT = {{1,0,0,0}, {0,1,0,0}, {0,0,0,1}, {0,0,1,0}}
HH = KroneckerProduct[H,H]
SI = KronekcerProduct[S,I]
IH = KroneckerProduct[ID,H]
CT = CNOT
XX = KroneckerProduct[X,X]
Phi0 = {1,0,0,0}
```

```
HH.XX.IH.CT.IH.XX.HH.SI.IH.CT.IH.SI.HH.Phi0


Out = {0,0,-1,0}
```

B.7 Experiment 4.6 *Mathematica* Notebook.

```
H = 1/(2^(1/2))*{{1,1}, {1,-1}}
ID = {{1,0}, {0,1}}
S = {{1,0}, {0,I}}
X = {{0,1}, {1,0}}
CNOT = {{1,0,0,0}, {0,1,0,0}, {0,0,0,1}, {0,0,1,0}}
HH = KroneckerProduct[H,H]
IH = KroneckerProduct[ID,H]
CT = CNOT
XX = KroneckerProduct[X,X]
Phi0 = {1,0,0,0}

HH.XX.IH.CT.IH.XX.HH.IH.CT.IH.HH.Phi0


Out = {0,0,0,-1}
```

B.8 Reversing a *CNOT* Gate *Mathematica* Notebook.

```
H = 1/(2^(1/2))*{{1,1}, {1,-1}}
CNOT = {{1,0,0,0}, {0,1,0,0}, {0,0,0,1}, {0,0,1,0}}
Phi0 = {0,1,0,0}
T1 = KroneckerProduct[H,H]
UCNOT = {{1,0,0,0}, {0,0,0,1}, {0,0,1,0}, {0,1,0,0}}

T1.CNOT.T1.Phi0
Out = {0,0,0,1}

UCNOT.Phi0
Out = {0,0,0,1}
```

B.9 Controlled-*S* Gate

```
T = {{1,0}, {0,e^(I*PI/4)}}
TA = {{1,0}, {0,e^(-I*PI/4)}}
ID = {{1,0}, {0,1}}
CNOT = {{1,0,0,0}, {0,1,0,0}, {0,0,0,1}, {0,0,1,0}}
T1 = KroneckerProduct[T,ID]
T2 = CNOT
T3 = KroneckerProduct[ID,TA]
T4 = CNOT
T5 = KroneckerProduct[ID,T]

T5.T4.T3.T2.T1
```

$$\text{Out} = \left\{\{1,0,0,0\}, \{0,1,0,0\}, \{0,0,1,0\}, \{0,0,0,e^{\frac{i\pi}{2}}\}\right\}$$

B.10 Experiment 4.8 *Mathematica* Notebook.

```
H = 1/(2^(1/2))*{{1,1}, {1,-1}}
ID = {{1,0}, {0,1}}
CNOT = {{1,0,0,0}, {0,1,0,0}, {0,0,0,1}, {0,0,1,0}}
CNOT3 = {{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}, {0,0,0,0,0,0,0,1},
{0,0,0,0,0,0,1,0}}
Phi0 = {1,0,0,0,0,0,0,0}
T1 = KroneckerProduct[X,ID,ID]
T2 = CNOT3
T3 = KroneckerProduct[H,ID,H]
T4 = CNOT3
T5 = T4
T6 = CNOT3
T7 = KroneckerProduct[ID, CNOT]
T8 = KroneckerProduct[ID,H,H]
T9 = T8
T10 = T9
T11 = T10
```

```
T12 = CNOT3
T13 = T4
T14 = CNOT3
T15 = T4
T16 = CNOT3

T16.T15.T14.T13.T12.T11.T10.T9.T8.T7.T6.T5.T4.T3.T2.T1.Phi0

Out = {0,0,1,0,0,0,0,0}
```

B.11 Experiment 4.10 *Mathematica* Notebook.

```
H = 1/(2^(1/2))*{{1,1}, {1,-1}}
ID = {{1,0}, {0,1}}
CNOT = {{1,0,0,0}, {0,1,0,0}, {0,0,0,1}, {0,0,1,0}}
CNOT3 = {{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}, {0,0,0,0,0,0,0,1},
{0,0,0,0,0,0,1,0}}
U1 = {{1,0}, {0, e^(I*PI/4}}
NU1 = {{1,0}, {0, e^(-I*PI/4}}
U2 = {{1,0}, {0, e^(I*PI/8}}
NU2 = {{1,0}, {0, e^(-I*PI/8}}
T1 = KroneckerProduct[H,ID,ID]
T2 = KroneckerProduct[U1,ID,ID]
T3 = KroneckerProduct[CNOT,ID]
T4 = KroneckerProduct[ID,NU1,ID]
T5 = T3
T6 = KroneckerProduct[ID,U1,ID
T7 = KroneckerProduct[ID,H,ID]
T8 = KroneckerProduct[U2,ID,ID]
T9 = CNOT3
T10 = KroneckerProduct[ID,ID,NU2]
T11 = T9
T12 = KroneckerProduct[ID,ID,U2]
T13 = KroneckerProduct[ID,U1,ID]
```

```
T14 = KroneckerProduct[ID,CNOT]
T15 = KroneckerProduct[ID,ID,NU1]
T16 = T14
T17 = KroneckerProduct[ID,ID,U1]
T18 = KroneckerProduct[ID,ID,H]
```

In[27]:=
```
T18.T17.T16.T15.T14.T13.T12.T11.T10.T9.T8.T7.T6.T5.T4.T3.T2.T1
```

Out[27]=

$$\left\{\left\{\frac{1}{2\sqrt{2}}, \frac{1}{2\sqrt{2}}, \frac{1}{2\sqrt{2}}, \frac{1}{2\sqrt{2}}, \frac{1}{2\sqrt{2}}, \frac{1}{2\sqrt{2}}, \frac{1}{2\sqrt{2}}, \frac{1}{2\sqrt{2}}\right\},\right.$$
$$\left\{\frac{1}{2\sqrt{2}}, -\frac{1}{2\sqrt{2}}, \frac{1}{2\sqrt{2}}, -\frac{1}{2\sqrt{2}}, \frac{1}{2\sqrt{2}}, -\frac{1}{2\sqrt{2}}, \frac{1}{2\sqrt{2}}, -\frac{1}{2\sqrt{2}}\right\},$$
$$\left\{\frac{1}{2\sqrt{2}}, \frac{e^{\frac{i\,PI}{2}}}{2\sqrt{2}}, -\frac{1}{2\sqrt{2}}, -\frac{e^{\frac{i\,PI}{2}}}{2\sqrt{2}}, \frac{1}{2\sqrt{2}}, \frac{e^{\frac{i\,PI}{2}}}{2\sqrt{2}}, -\frac{1}{2\sqrt{2}}, -\frac{e^{\frac{i\,PI}{2}}}{2\sqrt{2}}\right\},$$
$$\left\{\frac{1}{2\sqrt{2}}, -\frac{e^{\frac{i\,PI}{2}}}{2\sqrt{2}}, -\frac{1}{2\sqrt{2}}, \frac{e^{\frac{i\,PI}{2}}}{2\sqrt{2}}, \frac{1}{2\sqrt{2}}, -\frac{e^{\frac{i\,PI}{2}}}{2\sqrt{2}}, -\frac{1}{2\sqrt{2}}, \frac{e^{\frac{i\,PI}{2}}}{2\sqrt{2}}\right\},$$
$$\left\{\frac{1}{2\sqrt{2}}, \frac{e^{\frac{i\,PI}{4}}}{2\sqrt{2}}, \frac{e^{\frac{i\,PI}{2}}}{2\sqrt{2}}, \frac{e^{\frac{3\,i\,PI}{4}}}{2\sqrt{2}}, -\frac{1}{2\sqrt{2}}, -\frac{e^{\frac{i\,PI}{4}}}{2\sqrt{2}}, -\frac{e^{\frac{i\,PI}{2}}}{2\sqrt{2}}, -\frac{e^{\frac{3\,i\,PI}{4}}}{2\sqrt{2}}\right\},$$
$$\left\{\frac{1}{2\sqrt{2}}, -\frac{e^{\frac{i\,PI}{4}}}{2\sqrt{2}}, \frac{e^{\frac{i\,PI}{2}}}{2\sqrt{2}}, -\frac{e^{\frac{3\,i\,PI}{4}}}{2\sqrt{2}}, -\frac{1}{2\sqrt{2}}, \frac{e^{\frac{i\,PI}{4}}}{2\sqrt{2}}, -\frac{e^{\frac{i\,PI}{2}}}{2\sqrt{2}}, \frac{e^{\frac{3\,i\,PI}{4}}}{2\sqrt{2}}\right\},$$
$$\left\{\frac{1}{2\sqrt{2}}, \frac{e^{\frac{3\,i\,PI}{4}}}{2\sqrt{2}}, -\frac{e^{\frac{i\,PI}{2}}}{2\sqrt{2}}, -\frac{e^{\frac{5\,i\,PI}{4}}}{2\sqrt{2}}, -\frac{1}{2\sqrt{2}}, -\frac{e^{\frac{3\,i\,PI}{4}}}{2\sqrt{2}}, \frac{e^{\frac{i\,PI}{2}}}{2\sqrt{2}}, \frac{e^{\frac{5\,i\,PI}{4}}}{2\sqrt{2}}\right\},$$
$$\left.\left\{\frac{1}{2\sqrt{2}}, -\frac{e^{\frac{3\,i\,PI}{4}}}{2\sqrt{2}}, -\frac{e^{\frac{i\,PI}{2}}}{2\sqrt{2}}, \frac{e^{\frac{5\,i\,PI}{4}}}{2\sqrt{2}}, -\frac{1}{2\sqrt{2}}, \frac{e^{\frac{3\,i\,PI}{4}}}{2\sqrt{2}}, \frac{e^{\frac{i\,PI}{2}}}{2\sqrt{2}}, -\frac{e^{\frac{5\,i\,PI}{4}}}{2\sqrt{2}}\right\}\right\}$$

B.12 QFT from [1] *Mathematica* Notebook.

```
H = 1/(2^(1/2))*{{1,1}, {1,-1}}
ID = {{1,0}, {0,1}}
CNOT = {{1,0,0,0}, {0,1,0,0}, {0,0,0,1}, {0,0,1,0}}
CR2 = {{1,0,0,0}, {0,1,0,0}, {0,0,1,0}, {0,0,0,I}}
CR3 = {{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,e^((PI*I)/4),0,0},
{0,0,0,0,0,0,1,0}, {0,0,0,0,0,0,0,e^((PI*I)/4)}}
SWAP = {{1,0,0,0,0,0,0,0}, {0,0,0,0,1,0,0,0}, {0,0,1,0,0,0,0,0}
{0,0,0,0,0,0,1,0}, {0,1,0,0,0,0,0,0}, {0,0,0,0,0,1,0,0},
{0,0,0,1,0,0,0,0}, {0,0,0,0,0,0,0,1}}
T1 = KroneckerProduct[H,ID,ID]
T2 = KroneckerProduct[CR2,ID]
T3 = CR3
T4 = KroneckerProduct[ID,H,ID]
T5 = KroneckerProduct[ID,CR2]
T6 = KroneckerProduct[ID,ID,H]
```

In[13]:= **T6.T5.T4.T3.T2.T1**

Out[13]=

$$\left\{\left\{\frac{1}{2\sqrt{2}}, \frac{1}{2\sqrt{2}}, \frac{1}{2\sqrt{2}}, \frac{1}{2\sqrt{2}}, \frac{1}{2\sqrt{2}}, \frac{1}{2\sqrt{2}}, \frac{1}{2\sqrt{2}}, \frac{1}{2\sqrt{2}}\right\},\right.$$

$$\left\{\frac{1}{2\sqrt{2}}, -\frac{1}{2\sqrt{2}}, \frac{1}{2\sqrt{2}}, -\frac{1}{2\sqrt{2}}, \frac{1}{2\sqrt{2}}, -\frac{1}{2\sqrt{2}}, \frac{1}{2\sqrt{2}}, -\frac{1}{2\sqrt{2}}\right\},$$

$$\left\{\frac{1}{2\sqrt{2}}, \frac{i}{2\sqrt{2}}, -\frac{1}{2\sqrt{2}}, -\frac{i}{2\sqrt{2}}, \frac{1}{2\sqrt{2}}, \frac{i}{2\sqrt{2}}, -\frac{1}{2\sqrt{2}}, -\frac{i}{2\sqrt{2}}\right\},$$

$$\left\{\frac{1}{2\sqrt{2}}, -\frac{i}{2\sqrt{2}}, -\frac{1}{2\sqrt{2}}, \frac{i}{2\sqrt{2}}, \frac{1}{2\sqrt{2}}, -\frac{i}{2\sqrt{2}}, -\frac{1}{2\sqrt{2}}, \frac{i}{2\sqrt{2}}\right\},$$

$$\left\{\frac{1}{2\sqrt{2}}, \frac{e^{\frac{i\,PI}{4}}}{2\sqrt{2}}, \frac{i}{2\sqrt{2}}, \frac{i\,e^{\frac{i\,PI}{4}}}{2\sqrt{2}}, -\frac{1}{2\sqrt{2}}, -\frac{e^{\frac{i\,PI}{4}}}{2\sqrt{2}}, -\frac{i}{2\sqrt{2}}, -\frac{i\,e^{\frac{i\,PI}{4}}}{2\sqrt{2}}\right\},$$

$$\left\{\frac{1}{2\sqrt{2}}, -\frac{e^{\frac{i\,PI}{4}}}{2\sqrt{2}}, \frac{i}{2\sqrt{2}}, -\frac{i\,e^{\frac{i\,PI}{4}}}{2\sqrt{2}}, -\frac{1}{2\sqrt{2}}, \frac{e^{\frac{i\,PI}{4}}}{2\sqrt{2}}, -\frac{i}{2\sqrt{2}}, \frac{i\,e^{\frac{i\,PI}{4}}}{2\sqrt{2}}\right\},$$

$$\left\{\frac{1}{2\sqrt{2}}, \frac{i\,e^{\frac{i\,PI}{4}}}{2\sqrt{2}}, -\frac{i}{2\sqrt{2}}, \frac{e^{\frac{i\,PI}{4}}}{2\sqrt{2}}, -\frac{1}{2\sqrt{2}}, -\frac{i\,e^{\frac{i\,PI}{4}}}{2\sqrt{2}}, \frac{i}{2\sqrt{2}}, -\frac{e^{\frac{i\,PI}{4}}}{2\sqrt{2}}\right\},$$

$$\left.\left\{\frac{1}{2\sqrt{2}}, -\frac{i\,e^{\frac{i\,PI}{4}}}{2\sqrt{2}}, -\frac{i}{2\sqrt{2}}, -\frac{e^{\frac{i\,PI}{4}}}{2\sqrt{2}}, -\frac{1}{2\sqrt{2}}, \frac{i\,e^{\frac{i\,PI}{4}}}{2\sqrt{2}}, \frac{i}{2\sqrt{2}}, \frac{e^{\frac{i\,PI}{4}}}{2\sqrt{2}}\right\}\right\}$$

List of References

[1] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information*. Cambridge University Press, 2015.

[2] "IBM unveils roadmap for commercial 'IBM Q' quantum systems," Mar 2017. Available: http://www-03.ibm.com/press/us/en/pressrelease/51740.wss

[3] R. Feynman, "Simulating physics with computers," *International Journal of Theoretical Physics*, vol. 21, no. 6, pp. 467,488, 1982.

[4] D. Deutsch and R. Jozsa, "Rapid solution of problems by quantum computation," *Proceedings: Mathematical and Physical Sciences*, vol. 439, no. 1907, pp. 553–558, 1992. Available: http://www.jstor.org/stable/52182

[5] L. K. Grover, "Quantum mechanics helps in searching for a needle in a haystack," *Phys. Rev. Lett.*, vol. 79, pp. 325–328, Jul 1997. Available: https://link.aps.org/doi/10.1103/PhysRevLett.79.325

[6] P. W. Shor, "Algorithms for quantum computation: discrete logarithms and factoring," in *Proceedings 35th Annual Symposium on Foundations of Computer Science*, Nov 1994, pp. 124–134.

[7] J. Temperton, "Got a spare $15 million? Why not buy your very own D-Wave quantum computer," Jan 2017. Available: http://www.wired.co.uk/article/d-wave-2000q-quantum-computer

[8] C. Q. Choi, "IBM simulates a 56-qubit machine," Oct 2017. Available: https://spectrum.ieee.org/tech-talk/computing/hardware/ibms-quantum-leap-simulates-56qubit-machine

[9] J. M. Gambetta, J. M. Chow, and M. Steffen, "Building logical qubits in a superconducting quantum computing system," Jan 2017. Available: https://www.nature.com/articles/s41534-016-0004-0

[10] IBM, "IBM Quantum Experience - Full User Guide." Available: https://quantumexperience.ng.bluemix.net/qx/tutorial?sectionId=full-user-guide&page=introduction

[11] S. Shankland, "IBM quantum computers will unleash weird science on business," Mar 2017. Available: https://www.cnet.com/news/ibm-quantum-computers-business-moores-law-qubit/

[12] T. Simonite, "Moore's law is dead. Now what?" Feb 2017. Available: https://www.technologyreview.com/s/601441/moores-law-is-dead-now-what/

[13] D. J. Bernstein, *Post-Quantum Cryptography* (Lecture notes in computer science Post-quantum cryptography). Berlin, Heidelberg: Springer Berlin Heidelberg, 2009.

[14] C. M. Nikias, "This is the most important tech contest since the space race, and America is losing," May 2018. Available: https://www.washingtonpost.com/opinions/this-is-the-most-important-tech-contest-since-the-space-race-and-america-is-losing/2018/05/11/7a4a4772-4e21-11e8-b725-92c89fe3ca4c_story.html

[15] J. Koch, T. M. Yu, J. Gambetta, A. A. Houck, D. I. Schuster, J. Majer, A. Blais, M. H. Devoret, S. M. Girvin, R. J. Schoelkopf, and et al., "Charge-insensitive qubit design derived from the Cooper pair box," *Physical Review A*, vol. 76, no. 4, Dec 2007.

[16] QISKit, "QISKit/IBMqx-backend-information." Available: https://github.com/QISKit/ibmqx-backend-information/blob/master/backends/ibmqx2/README.md

[17] P. A. M. Dirac, "A new notation for quantum mechanics," *Mathematical Proceedings of the Cambridge Philosophical Society*, vol. 35, no. 3, pp. 416–418, 1939.

[18] N. S. Yanofsky and M. A. Mannucci, *Quantum computing for computer scientists*. Cambridge University Press, 2013.

[19] J. H. Luscombe, *Thermodynamics*. CRC Press, Taylor Francis Group, 2018.

[20] A. Bérut, A. Arakelyan, A. Petrosyan, S. Ciliberto, R. Dillenschneider, and E. Lutz, "Experimental verification of Landauer's principle linking information and thermodynamics," *Nature*, vol. 483, no. 7388, pp. 187–9, Mar 08 2012. Available: http://libproxy.nps.edu/login?url=https://search.proquest.com/docview/963550044?accountid=12702

[21] S. Aaronson and D. Gottesman, "Improved simulation of stabilizer circuits," *Physical Review A*, vol. 70, no. 5, 2004.

[22] A. Barenco, C. H. Bennett, R. Cleve, D. P. DiVincenzo, N. Margolus, P. Shor, T. Sleator, J. A. Smolin, and H. Weinfurter, "Elementary gates for quantum computation," *Phys. Rev. A*, vol. 52, pp. 3457–3467, Nov 1995. Available: https://link.aps.org/doi/10.1103/PhysRevA.52.3457

[23] A. W. Cross, L. S. Bishop, J. A. Smolin, and J. M. Gambetta, "Open quantum assembly language," *arXiv preprint arXiv:1707.03429*, 2017.

[24] R. P. Feynman, *The Feynman lectures on physics*. Addison-Wesley, 1989 - 1963.

[25] A. Einstein, "On a heuristic point of view concerning the production and transformation of light," *Annalen der Physik*, pp. 1–18, 1905.

[26] S. Kocsis, B. Braverman, S. Ravets, M. J. Stevens, R. P. Mirin, L. K. Shalm, and A. M. Steinberg, "Observing the average trajectories of single photons in a two-slit interferometer," *Science*, vol. 332, no. 6034, pp. 1170–1173, 2011.

[27] R. Bach, D. Pope, S.-H. Liou, and H. Batelaan, "Controlled double-slit electron diffraction," *New Journal of Physics*, vol. 15, no. 3, p. 033018, 2013.

[28] A. Einstein, M. Born, and H. Born, *The Born-Einstein letters: Friendship, politics and physics in uncertain times*. Macmillan, 2005.

[29] A. Einstein, B. Podolsky, and N. Rosen, "Can quantum-mechanical description of physical reality be considered complete?" *Physical Review*, vol. 47, no. 10, p. 777–780, 1935.

[30] D. M. Greenberger, M. A. Horne, and A. Zeilinger, "Going beyond Bell's theorem," in *Bell's theorem, quantum theory and conceptions of the universe*. Springer, 1989, pp. 69–72.

[31] J. S. Bell, "On the Einstein-Podolsky-Rosen paradox," *Physics*, vol. 1, pp. 195–200, 1964.

[32] J. F. Clauser, M. A. Horne, A. Shimony, and R. A. Holt, "Proposed experiment to test local hidden-variable theories," *Phys. Rev. Lett.*, vol. 23, pp. 880–884, Oct 1969. Available: https://link.aps.org/doi/10.1103/PhysRevLett.23.880

[33] J.-W. Pan, D. Bouwmeester, M. Daniell, H. Weinfurter, and A. Zeilinger, "Experimental test of quantum nonlocality in three-photon Greenberger–Horne–Zeilinger entanglement," *Nature*, vol. 403, no. 6769, p. 515, 2000.

[34] IBM, "IBM Q Composer." Available: https://quantumexperience.ng.bluemix.net/qx/editor

[35] H. Jarlett, "Breaking data records bit by bit," Dec 2017. Available: https://home.cern/about/updates/2017/12/breaking-data-records-bit-bit

[36] P. W. Shor, "Introduction to quantum algorithms," in *Proceedings of Symposia in Applied Mathematics*, 2002, vol. 58, pp. 143–160.

[37] S. Beauregard, "Circuit for Shor's algorithm using 2n+ 3 qubits," *arXiv preprint quant-ph/0205095*, 2002.

[38] I. L. Markov and M. Saeedi, "Constant-optimized quantum circuits for modular multiplication and exponentiation," *arXiv preprint arXiv:1202.6614*, 2012.

[39] "Pauli matrices," May 2018. Available: https://en.wikipedia.org/wiki/Pauli_matrices#The_group_composition_law_of_SU(2)

[40] "Massively parallel quantum computer simulator," *Computer physics communications*, vol. 176, no. 2, pp. 121,136, 2007.

[41] C. Q. Choi, "IBM simulates a 56-qubit machine," Oct 2017. Available: https://spectrum.ieee.org/tech-talk/computing/hardware/ibms-quantum-leap-simulates-56qubit-machine

[42] "D-Wave announces upgrades to D-Wave 2000Q quantum computer," Nov 2017. Available: https://www.dwavesys.com/press-releases/d-wave-announcesupgrades-d-wave-2000q-quantum-computer

[43] "The D-Wave 2000Q quantum computer technology overview." Available: https://www.dwavesys.com/sites/default/files/D-Wave2000QTechCollateral_0117F.pdf

[44] S. Aaronson, "Insert D-Wave post here," Mar 2017. Available: https://www.scottaaronson.com/blog/?p=3192

[45] S. W. Shin, G. Smith, J. A. Smolin, and U. Vazirani, "How "quantum" is the D-Wave machine?" *arXiv preprint arXiv:1401.7087*, 2014.

[46] T. F. Rønnow, Z. Wang, J. Job, S. Boixo, S. V. Isakov, D. Wecker, J. M. Martinis, D. A. Lidar, and M. Troyer, "Defining and detecting quantum speedup," *Science*, vol. 345, no. 6195, pp. 420–424, 2014.

[47] J. Preskill, "Quantum computing in the NISQ era and beyond," *arXiv preprint arXiv:1801.00862*, 2018.

[48] T. Curcic, M. E. Filipkowski, A. Chtchelkanova, P. A. D'Ambrosio, S. A. Wolf, M. Foster, and D. Cochran, "Quantum networks: from quantum cryptography to quantum architecture," *ACM SIGCOMM Computer Communication Review*, vol. 34, no. 5, pp. 3–8, 2004.

[49] K. Michielsen, M. Nocon, D. Willsch, F. Jin, T. Lippert, and H. D. Raedt, "Benchmarking gate-based quantum computers," *Computer Physics Communications*, vol. 220, pp. 44 – 55, 2017. Available: http://www.sciencedirect.com/science/article/pii/S0010465517301935

[50] S. J. Devitt, "Performing quantum computing experiments in the cloud," *Phys. Rev. A*, vol. 94, p. 032329, Sep 2016. Available: https://link.aps.org/doi/10.1103/PhysRevA.94.032329

[51] Anonymous, “Quantum Programming Language,” Dec 2015. Available: https://www.quantiki.org/wiki/quantum-programming-language

[52] B. Omer, “Quantum programming in QCL,” *Master’s thesis, Institute of Information Systems Technical University of Viennna*, 2000.

THIS PAGE INTENTIONALLY LEFT BLANK

Initial Distribution List

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California