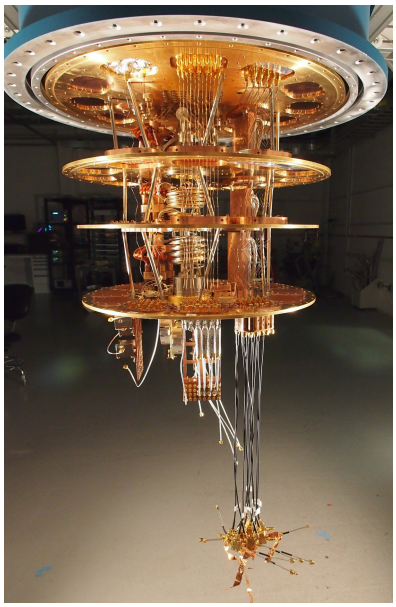
A Brief Introduction to Quantum Computing from the Perspective of Ladder Logic

Jerry Kensler

June 8, 2018



Abstract

Quantum Computing is an advanced topic, it suffers from a perception of complexity beyond what is reasonable for the actual subject matter. The intent of this paper is to soften that perception and bring the topic down to a less intimidating level. The primary targets of this paper are students currently enrolled in or freshly graduated from an electrical engineering program; however, any individual with a base knowledge in programming or digital logic should be able to gain some level of benefit.

Keywords: Quantum, QISKit, Computing, Ladder, Logic, QASM, Introduction

Figure 1: Photograph of Bristlecone courtesy of Google(CITE)

1 Introduction

Quantum physics as a whole is an exceptionally advanced topic and the applications, occasionally breaching into concepts such as particles that are both matter and antimatter[10] which may at first seem contradictory by their very nature. Add to this many common misleading analogies and the fact that "Qubit" may refer to any number of technologies, such as ion traps, superconducting qubits, or spin qubits and it is no wonder that people get confused. While quantum computing does use the underlying principles of quantum mechanics, it is still nothing more than a programming architecture. To put this statement in another context, an individual does not need to understand how to bias a transistor to be good at programming in C.

1.1 Use Case for Quantum Computing

Quantum computing is a Disruptive Emergent Technology, meaning, as far as industry is concerned, the implementation of these concepts have not only never been seen in any technology prior, but they also have the potential to revolutionize how things are done should these implementations prove successful. Quantum computing is not a magic bullet that can handle every problem; however, it does have the potential to be exceptionally good at problems that current technology has difficulty solving. Examples of such problems can be as varied as weather prediction, mathematical factoring [7], and even protein folding [11].

1.2 Relevance

First, advances in technology generally creates a higher standard of living, in the case of quantum computing, these advances may assist in finding the cure to debilitating diseases, reducing cost of living (CITE OR REPLACE), or even allowing for a faster, more secure means to transport data (Quantum Internet [5]).

Second is economics, the reality of the situation is that there aren't enough people who are skilled in this area to fit the demand. Meaning, not only are jobs available to those with the skills to fill the required roles, but due to the shortage of supply when compared to the demand, these jobs generally pay quite well. [4]

2 Background Concepts

This paper assumes the reader has some basic knowledge of programming and electronics. The content will primarily focus on the act of programming itself as well as the topics needed to get to that point. There are many topics which should be covered if one wishes to become skilled in quantum computing, most of which the reader will need to research independently; however, the sections below should provide enough context in order to provide a suitable starting point.

2.1 Removing Misleading Assumptions

The first and easily one of the most important concepts in quantum computing is to understand that the general public and most media 'experts' do not work in the field of quantum computing. Therefore, it is important to note that many of the common assumptions can be misleading or plain wrong due to lack of context or the background knowledge required to properly articulate the concepts at hand.

One prime example of this is Erwin Schrödinger and the concept of Schrödinger's cat. To preface, this is not at all meant to downplay or discredit his work, merely to illustrate that without proper context even an otherwise correct statement from a Nobel laureate can be misconstrued.

To briefly summarize, Schrödinger's cat is a brilliant thought experiment meant to illustrate a potential paradox present within the Copenhagen interpretation of quantum mechanics. This thought experiment was meant to highlight the bizarre nature of EPR (Einstein, Podolsky, Rosen), or superposition states. This is demonstrated via a cat, radioactive particle, flask of poison, and a Geiger counter being sealed in a theoretical box. Should the counter detect radioactivity, the flask would be broken causing the cat to die immediately. Under the Copenhagen interpretation of quantum mechanics, this cat should be considered simultaneously both alive and dead, however, looking into the sealed box will reveal either a very scared cat, or a testament to animals killed in the name of science.

In context, this is meant to demonstrate the concepts of complex, superimposed states collapsing upon measurement. For better or worse, this concept of a not dead, yet dead cat in a box has spread like wildfire, it has become the cornerstone of what the public sees as quantum computing. This has allowed the idea that the quantum state is both 1 and 0 to become the very first thing that anyone learns in regards to quantum computing. While this statement is not technically wrong, it is fundamentally incomplete. In many ways, it's similar to the question "which came first: the chicken, or the egg?", the answer to which can only be something along the lines of 'invalid question', as it not only fails to define what denotes the term chicken, but it also leads the individual being asked to assume the term 'egg' is specifically in the context of a chicken egg.

In reality, a better way to think of this is through vectors and complex numbers. The value is in fact both 1 and 0, but it's that way not because it is two things in a binary sense, but because it is a complex mixing of both. To put this back into cat analogies, let's take the premise back to the start. There is a cat, that cat is now infected with a zombie pathogen. For all purposes, the cat is neither alive, nor dead, as it cannot cleanly fit into either category, yet it does have many of the defining characteristics that comprise both. From here, the cat is injected with an antidote, being a rushed marvel of medicine meant to save the feline race, it will near instantly result in either a complete cure or certain death for our kitten subject. For this analogy, the zombified state represents the ability of qubits to be in a mixed state, the antidote representing the act of measuring this mixed state and thus collapsing it into a binary value.

Signed Decimal	Binary (IEEE 754*)	Balanced Ternary
0	0	0
3	11	10
5	101	+ 0 -
-254	11000011011111100000000000000000*	- 0 0 - + -

Table 1: Comparison table showing equivalent numbers in different display forms

2.2 Balanced Ternary

Balanced Ternary is not, strictly speaking a part of quantum computing, so with that knowledge, it may seem like a strange item to include as background information when discussing the subject. However, given this author's personal experience, it is nearly ideal to serve not only as a bridge to more advanced concepts but also as a demonstration of why said concepts are important.

To do this, a brief nod to what numbers are at their core is needed. It's pretty obvious to most, but numbers are nothing more or less than a way to categorize quantity. This concept of quantity is important as it is independent of stylistic choices such as base or notation. In programming, binary is frequently used as it reflects the power state of transistors within the register. For most general cases, this works quite well allowing for any number of tricks, or bit-hacks, to be employed to speed things up. However, unsigned binary does have an Achilles' heel in that due to its basis in positive numbers, there is no simple way to show when a number is negative on hardware [1]. To do so, one needs to employ a tactic such as the IEEE 754 standard, adding much more complexity than would otherwise be necessary. One way to circumvent this issue is to use a balanced number system such as balanced ternary.

These systems work by designating symbols as pre-signed. In the case of balanced ternary, the symbols are designated as "0; +; -" and they represent "$3^{(s*n)}$" with "n" being the index number starting at zero from right to left and "s" being the sign as denoted by the "+" for positive, "-" for negative, or "0" for null quantity [2]. From there it is a simple matter of adding the positive and negative values up in order to get the end result (see Table 1, for example and comparison between bases).

2.3 Vectors (Explain Each New symbol)

A proper discussion of quantum computing cannot be had without discussing vectors and their function in understanding the circuit. This will be discussed more in a later section (4.1.1); however, the shorthand version is as follows. Due to the fact that quantum computers operate on qubits rather than classical (binary) bits, a quantum computer is effectively nothing more than a very com-

[1] Note: the "-" symbol seen in both binary and standard decimal is not a representation of quantity, it's more akin to a shorthand for the operation of 0 - Quantity, and thus it's fairly difficult to represent in hardware

[2] This notion of sign polarity being a property of quantity will become relevant in later sections (3.3), but for now, it is acceptable to use this idea as a base case.

plex probabilistic vector equation. To be specific, this state space $\mathcal{H}$, is not just vector space but a Hilbert space, meaning it has a positive-definite Hermitian inner product $\langle \cdot | \cdot \rangle$. This has quite a few implications and entire papers can be written on the vector space alone, but in simple terms what it means is that the sum of the probabilities of a quantum computer being in any given state post measurement will always be 1.

2.3.1 Notation

In quantum computing, and quantum mechanics as a whole for that matter the standard notation is Bra-Ket notation, also known as Dirac notation. Example: $\langle \phi | \psi \rangle$. The right part, called ket, is typically represented as a column vector and written as $|\psi\rangle$. The left part, called bra, is the Hermitian conjugate of the ket with the same label, is typically represented as a row vector and written as $\langle \phi |$.

This may seem like a bit much for those whom have never worked with this sort of thing before, but the following example should help clarify things.
$|\psi\rangle = \alpha |H\rangle + \beta |V\rangle$
What the above example notates is the simplest meaningful version of a quantum system that can be in two possible states, for example the polarization of a photon. When measured, the system will collapse into either the horizontal state $|H\rangle$ or the vertical state $|V\rangle$, until that moment it exists in a superposition of the two. To find the probability of the system collapsing into a given state upon measurement, simply take the multiplier in front of the desired ket and square it. For instance, in the case of finding the probability of a vertical state the answer would be β^2.

Back to the previous section, the fact that the sum of the probability of all possible states is equal to 1 was brought up. This can be notated as:
$\sum \psi_n^* \psi_n = 1$
This will be discussed in more detail in the qubit section (4.1.1) in regards to what are called bloch spheres.

3 Advanced Concepts

The following sections are more advanced concepts moving towards the target goal of quantum circuits. These sections assume that the reader understands all prior sections and has completed further reading when necessary.

3.1 Reversible Logic Gates

A more advanced concept that is still core to quantum computing is the idea of reversible logic gates. Normally when it comes to digital logic inputs and outputs are decoupled. Consider the digital logic circuit as seen in Figure 2. In this circuit, there are three inputs and a single output, even knowing the gate layout and the end value of f_1, the best that could be done for determining the input values is narrowing it to a range of possibilities as all the values have been combined into one, irreversible output. Now consider the reversible logic

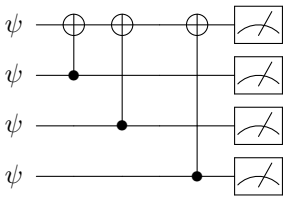


Figure 3: Reversible Logic Circuit

circuit as seen in Figure 3, while the notation may be unfamiliar[3], each input has its own unique output. Due to this, so long as one knows what gates are in the circuit and what the output values are, the input values can be derived relatively easily.

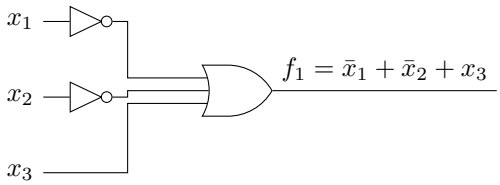


Figure 2: Digital Logic Circuit

3.2 PLC Ladder Logic images and content required

3.3 Probability Current(Placeholder)

In the previous section involving balanced ternary it was mentioned that the concept of quantities innately having a sign polarity would be relevant later, this section is that aforementioned later. Fair warning, this section will get very math heavy.

[3]This notation is that of a simple quantum ladder logic circuit, the elements of which will be covered in section 4.1

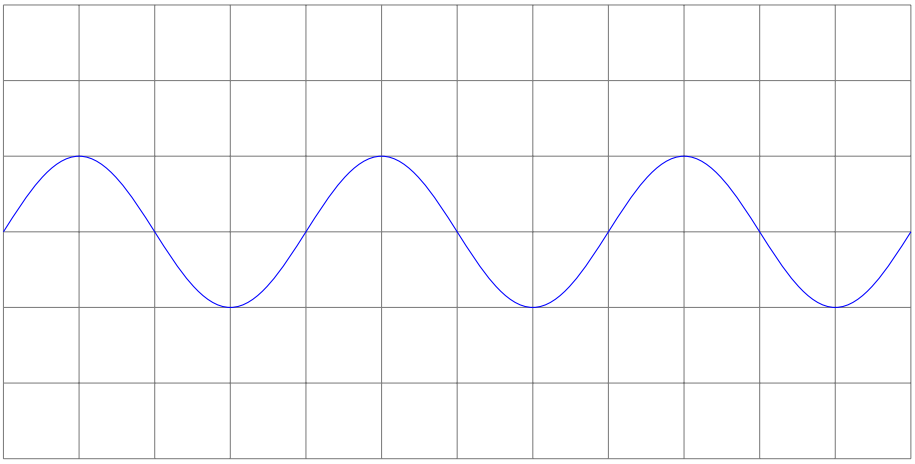


Figure 4: Probability Density Placeholder

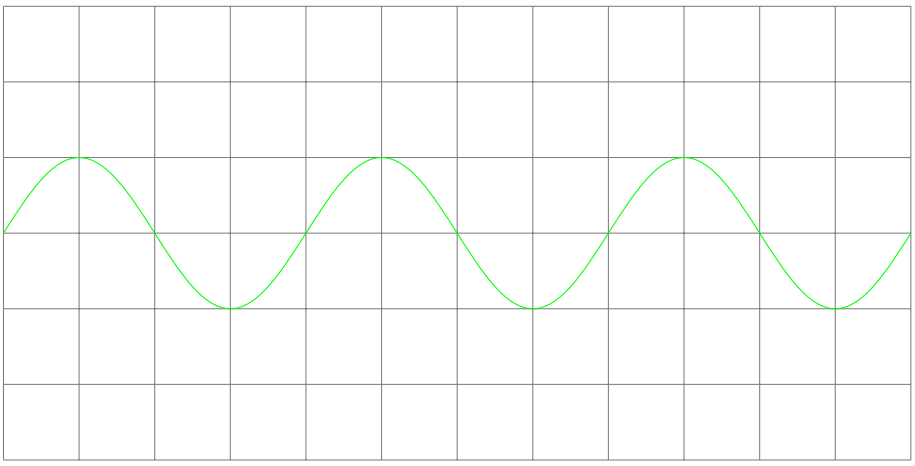


Figure 5: Probability Current Placeholder

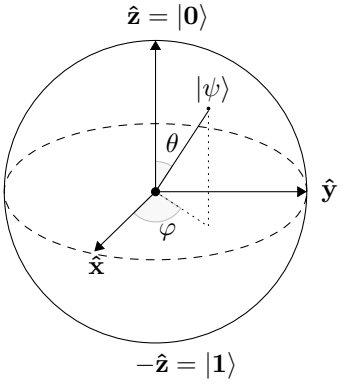


Figure 6: Bloch Sphere

4 Quantum Coding

This section deals in the various coding languages utilized in programming quantum computers. As these languages are still in active development, expect this section to change frequently.

4.1 Quantum Ladder Logic

4.1.1 Qubits

Into the meat of the matter, qubits are to quantum computers what register bits are to a classical computer. A qubit is a two-state quantum mechanical system such as photon polarization or the spin of a trapped ion.

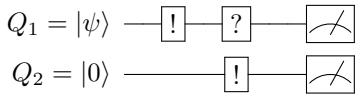
Bloch Sphere See Figure 6 for visual. A Bloch sphere is a geometrical representation of the pure state space of a two level quantum mechanical system (qubit).

Since the total probability of the system has to be one: $\langle\psi|\psi\rangle = 1$ this gives us the constraint required in order to write $|\psi\rangle$ using the following representation.
$|\psi\rangle = \cos\left(\frac{\theta}{2}\right)|0\rangle + e^{i\phi}\sin\left(\frac{\theta}{2}\right)|1\rangle = \cos\left(\frac{\theta}{2}\right)|0\rangle + (cos(\phi) + i\sin(\phi))\sin\left(\frac{\theta}{2}\right)|1\rangle$
Where $0 \leq \theta \leq \pi$.

All of that is a bit of a mouth-full, so to speak, but for most practical purposes, the Bloch sphere is one of the best tools to visually understand and debug a quantum circuit on the qubit level when it is an option that is available.

4.1.2 Circuit Layout

Similar to PLC Ladder logic (See section 3.2 for details), quantum circuits displayed in ladder logic generally follow a fairly well established layout.



Unless otherwise noted, all quantum circuits are treated as going from left to right. Each row is a unique qubit and the line going across the row is the quantum wire [4]. In the above circuit, there are two qubits, the top qubit has been initialized to $|\psi\rangle$, or an arbitrary quantum state, below it the bottom qubit has been initialized to $|0\rangle$.

For theoretical purposes, most qubits in quantum circuits are notated as having an initial state of $|\psi\rangle$, this is to show that the algorithm these qubits are within works regardless of the input state of these qubits; however, when it comes to real-world applications, most qubits will start in a state of either $|1\rangle$ or $|0\rangle$ as it eliminates unnecessary variables when performing calculations.

After initialization, the qubits continue along the wire from left to right in parallel. Functionally, this is akin to having a vertical ruler with enough length to cover every row and a thin enough width to only take up a single column. As this ruler advances, the operations under it will be evaluated. Any 'bare' wire will be evaluated as a no-op or no operation, and any gates will be evaluated for the time column they are contained in.

In the case of our example circuit above, after the initial states are set, the order of operations is as follows [5]:

$$\boxed{Q_1 = no-op; Q_2 = no-op} \boxed{\rightarrow} \boxed{Q_1 = Gate"!"; Q_2 = no-op} \boxed{\rightarrow} \boxed{Q_1 = no-op; Q_2 = no-op}$$
$$\boxed{\rightarrow} \boxed{Q_1 = Gate"?"; Q_2 = Gate"!"} \boxed{\rightarrow} \boxed{Q_1 = no-op; Q_2 = no-op} \boxed{\rightarrow}$$
$$\boxed{Q_1 = Measure; Q_2 = Measure}$$

This methodology of parallelism is a core property inherent to how a quantum computer processes data at a fundamental level. Thus, regardless of how a quantum computer is programmed, most quantum circuits can be approximated in this manor.

[4] It is important to note here that all operations, including a 'blank' wire are 'gates' so to speak. The implications of this will be discussed in the section regarding quantum gates (Section: 4.1.3)

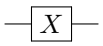
[5] For this example, the $\rightarrow$ symbol is being used to represent the advancement of time between columns/steps

4.1.3 Quantum Gates

Much like a digital logic gate operates on an individual or set of classical bits, a Quantum gate operates on individual or sets of qubits. Given that each qubit is effectively a matrix, quantum gates are matrices which are used to modify the vector space.

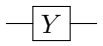
Single-Qubit Gates Among the single qubit gates, the most well known and frequently used are the Pauli gates, the Hadamard transform, the Phase gate, and the $\frac{\pi}{8}$ gate (denoted as T). These gates are as follows:

Pauli X



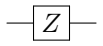
$$\theta_1 = \theta_X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

Pauli Y



$$\theta_2 = \theta_Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}$$

Pauli Z

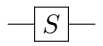


$$\theta_3 = \theta_Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

Hadamard Transform

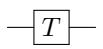
$$\theta_4 = \theta_H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

Phase Gate



$$\theta_5 = \theta_S = \begin{bmatrix} 1 & 0 \\ 1 & i \end{bmatrix}$$

$\frac{\pi}{8}$ or T gate



$$\theta_6 = \theta_T = \begin{bmatrix} 1 & 0 \\ 1 & exp(\frac{i\pi}{4}) \end{bmatrix}$$

Multi-Qubit Gates In addition to single qubit gates, most quantum circuits also utilize multi-qubit gates. There are many different types of these, but the most basic case is the controlled gate.

Control

Advanced Controls

4.2 Quantum Assembly Language

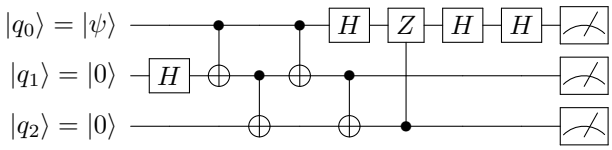
Quantum Assembly Language [2], also known as QASM or OpenQASM is a coding back end that is widely used throughout industry. It was developed to solve many of the coding issues that had been plaguing the architecture due to lack of a widely accepted standard. As well as being part of IBM's Quantum Experience interface, its utilization can be seen in projects such as QISKit as well as many smaller projects.

In terms of programming, QASM is relatively straightforward; there are only two types of storage, classical registers, and quantum registers. These registers are nothing more than one dimensional arrays of bits or qubits respectively and can then have gates applied to them which are unitary subroutines. Below is an example of QASM in use as well as the quantum ladder logic equivalent.

```
include "qelib1.inc";

def c-Z,1,'Z'

qubit q0, $\psi$
qubit q1, 0
qubit q2, 0

H q1
cnot q0,q1
cnot q1,q2
cnot q0,q1
cnot q1,q2
H q0
```

```
c-Z q2,q0
H q0
H q0

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

Which Equates to the following code in Ladder Logic



The circuit and code given above is for a known simplification of quantum teleportation [1]. (Repeated Content?)

5 Experiment (Provide screencaps for each)

This section contains information on experiments performed on both real world quantum computers as well as simulations and approximations of quantum computers.

5.1 IBM Quantum Experience

When this project first began, IBM's Quantum Experience (IBMQE) was still very young. It was and still is revolutionary in that it gave registered users the ability to program and test code on real quantum computers. The next sections will cover details in regards to the primary methods of programming through this interface.

5.1.1 Ladder logic

For most users of IBMQE this is the primary thing they'll see. It's a standard ladder logic interface and because of this, it is fairly intuitive. With that said, because IBMQE is running on real hardware, it suffers from the restrictions of said hardware. Qubits here are implemented physically via Pi Josephson Junctions, effectively a superconducting anharmonic oscillator. This means that the frequency a qubit is oscillating at determines what qubits it can interact with. Speaking from experience, to a new user that is extremely confusing if one isn't given an explanation. The interface also lacks many basic multi-qubit gates due to their physical implementation not being possible at current time.

IBMQE can move into a pure simulation mode, but it lacks the live feedback to properly make that mode seem responsive. Where it does excel is in the next section, Quantum Assembly language (QASM).

5.1.2 QASM

In addition to the ability to program in ladder logic, IBMQE allows users to code their circuits in QASM. This feature is extremely powerful as users can not only code in ladder logic or QASM, but they can also flip between them depending on what view best suits their needs. A skilled user can easily define custom gates based on matrices should the standard pauli gates not be sufficient.

5.2 Quirk

5.2.1 Definition

Quirk is an idealized, online drag-and-drop quantum circuit simulator. This simulator is hosted on Craig Gidney's block Algorithmic Assertions[17]. It is open source and due to the fact that he works as a part of Google's Quantum AI Team, it is owned by Google.

5.2.2 Pros

Among all currently available quantum simulation interfaces, this one stands out from the crowd. It is responsive, accessible online without having to download software and extremely intuitive. All in all, it's an excellent place to begin learning about quantum computing, especially given that the blog hosting it has a plethora of articles about the inner workings of many quantum algorithms.

5.2.3 Cons

For all the Quirk interface brings to the table, there is one very important thing that it does not do. Quirk is incapable of interfacing with a live quantum computer via API or otherwise; moreover, given the fact that circuits are stored as URL's to be generated, this severely limits how well the code can store as all it takes to lose a code is accidentally overwriting the clipboard or text document that said circuit is stored within.

(Create a table comparing the various simulation and programming interfaces)

6 Results and Interpretation

6.1 Example Source Code

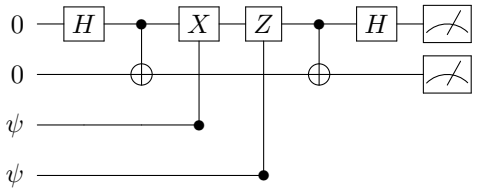
The following are examples of code that was run live on a quantum computer, specifically on IBM Q 5 Yorktown and related devices.

6.1.1 Superdense Coding

In this section, the code provided is in both QASM and ladder logic. It is worth noting that there is a discrepancy between the ladder logic and the QASM in regards to the cnot and controlled z gates. This is due to the fact that at this time, there is no direct hardware implementation of a controlled z gate, thus for simplicity's sake the QASM code assumes non-existent control bits of 1. This assumption can be changed to a control bit of zero by commenting out these gates. See Figure 7 in the Appendix for an example of program output on IBM's interface.

```
include "qelib1.inc";
qreg q[5];
creg c[5];
h q[0];
cx q[0],q[1];
x q[0];
z q[0];
cx q[0],q[1];
h q[0];
measure q[0] -> c[0];
measure q[1] -> c[1];
```

Which Equates to the following code in Ladder Logic



This may not seem like much at first glance but this small circuit shows many of the key fundamentals of quantum computing. Within this small circuit are important concepts such as superposition, entanglement, and even advanced controls. These all coalesce to enable the top qubit to carry two bits of information which are controlled by the bottom two gates. From there, it can be referenced to the qubit that was previously entangled with it in order to determine what the net change is and output the results via measurement.

6.2 Analysis (Provide list of original goals)

Given that the goal of this project was to teach myself the basics of quantum computing and provide a path for other undergraduates to do the same, this project was a resounding success. In the scope of this project, I taught myself the following languages: Quantum Ladder Logic, Quantum Assembly Language, Q# (Microsoft's Quantum Computing language), LaTeX, CSS, and how to work with QISKit a specific coding API for quantum computers which is written in Python.

In addition to the coding languages this project enabled me to pursue, I was able to meet with multiple professionals in the industry of Quantum Computing, from PhD's in quantum physics, to up and coming startups. (Nod to

companies and people perspectives, do not give individual's names) Moreover, the knowledge and source material required for this project to happen has also been requested by multiple professors at varied levels of teaching. Partly in response to this, I have set up this paper to be hosted on GitHub so that anyone can provide feedback, allowing this document to stay relevant.

6.3 Future Plans

For future plans, this project is something I intend to continuously work on, with the eventual goal of including in depth tutorials for all extant programming languages in quantum computing. At this point it has already been multiple years since this project was originally started and in all likelihood, this will be maintained and improved for years to come. It would be beneficial to add a few industry professionals to oversee the document, ensuring accuracy by catching errors that I would not notice. Additionally, in time it should be possible to incorporate more features into this paper whether via source code or the output PDF itself, separating out functions such as the Bloch sphere into its own library that anyone would be able to use.

7 Appendix

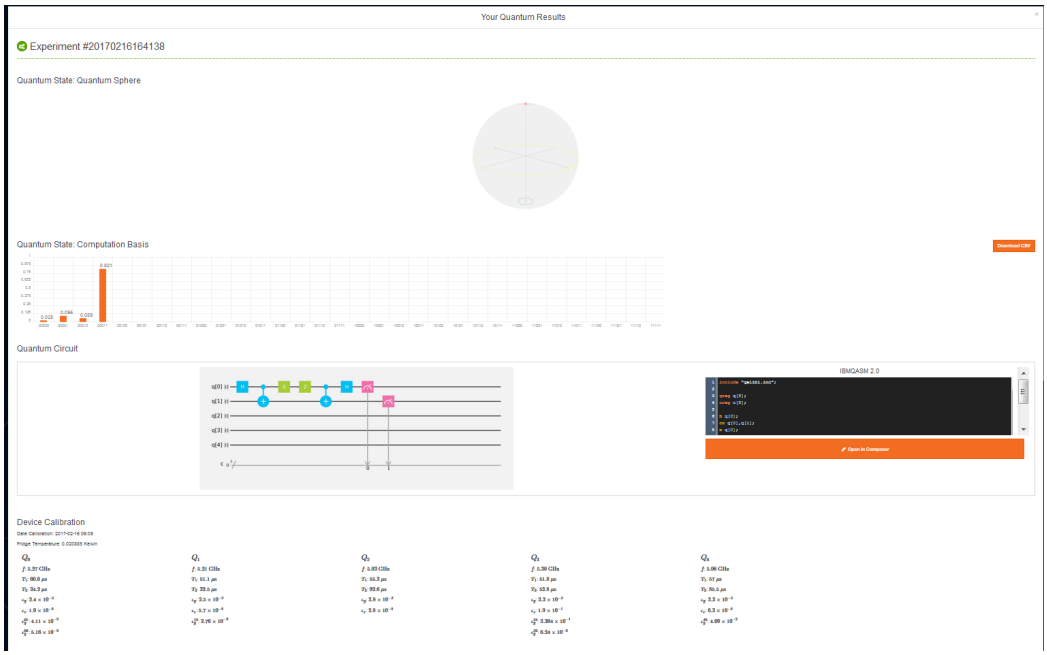


Figure 7: Example IBM Quantum Experience Results

References

[1] Nielson M.A., Chuang I.L. *"Quantum Computation and Quantum Information.* Cambridge University Press, Cambridge. 10th Anniversary Edition. ISBN 978-1-107-00217-3

[2] Andrew W. Cross, Lev S. Bishop, John A. Smolin, Jay M. Gambetta. *"Open Quantum Assembly Language"*. ARXIV July 12 2017, eprint arXiv:quant-ph/1707.03429v2. Accessed November 15 2017. https://arxiv.org/pdf/1707.03429.pdf

[3] University of New Mexico. *"Q-circuit Tutorial"* ARXIV, August 24 2004, eprint arXiv:quant-ph/0406003. Bryan Eastin, Steven T. Flammia, Department of Physics and Astronomy. Accessed May 20 2018. https://arxiv.org/pdf/quant-ph/0406003.pdf

[4] Google Quantum AI Laboratory. *"Commercialize quantum technologies in five years"*. March 03 2017. Masoud Mohseni, Peter Read, Hartmut Neven, Sergio Boxio, Vasil Denchev, Ryan Babbush, Austin Fowler, Vadim Smelyanskiy, John Martinis. Accessed May 06 2018. https://www.nature.com/news/commercialize-quantum-technologies-in-five-years-1.21583

[5] Castelvecchi, Davide. *"The quantum internet has arrived (and it hasn't)"*. February 14 2018, Nature. Accessed February 20 2018. https://www.nature.com/articles/d41586-018-01835-3

[6] Aaronson, Scott. *"Shor, I'll do it"*. February 24 2007, Shtetl-Optimized. Accessed December 02 2016. https://www.scottaaronson.com/blog/?p=208

[7] Shor, Peter W. *"Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer"*. ARXIV, August 30 1995, eprint arXiv:quant-ph/9508027. Accessed August 30 2017. https://arxiv.org/abs/quant-ph/9508027

[8] Forum Users. *"Prime Numbers: In Layman's Terms, How does Shors Algorithm Work?"*, October 9 2012. Cryptography Stack Exchange. StackExchange, Accessed March 4 2017. https://crypto.stackexchange.com/questions/3932/in-laymans-terms-how-does-shors-algorithm-work

[9] Richard Newrock, *"What are Joesphson Junctions? How do they work?"* Scientific American

[10] Moskowitz, Clara. *"New Particle Is Both Matter and Antimatter"*. October 2 2014, Scientific American. Accessed April 4 2018. https://www.scientificamerican.com/article/majorana-particle-matter-and-antimatter/

[11] Brumfiel, Geoffrey. *"D-Wave quantum computer solves protein folding problem"*. August 17 2012. Nature. Accessed August 1 2017. http://blogs.nature.com/news/2012/08/d-wave-quantum-computer-solves-protein-folding-problem.html

[12] Shor, Peter. *"Quantum Computation".* MIT OpenCourseware. Massachusetts Institute of Technology, 2003,18.435J / 2.111J / ESD.79J, https://ocw.mit.edu/courses/mathematics/18-435j-quantum-computation-fall-2003/

[13] O'Donnell, Ryan. Wright, John. *"Quantum Computation and Information".* Carnegie Mellon University, 2015, 15-859BB, https://www.cs.cmu.edu/ odonnell/quantum15/

[14] Jordan, Stephan. *"Quantum Algorithm Zoo".* National Institute of Standards and Technology (NIST), https://math.nist.gov/quantum/zoo/

[15] QISKit. *"QISKit."* https://github.com/QISKit. Accessed: February 14 2018

[16] Wooten, James. *"Using a Simple Puzzle Game to Benchmark Quantum Computers".* Medium, January 16 2018. https://medium.com/@decodoku/understanding-quantum-computers-through-a-simple-puzzle-game-a290dde89fb2

[17] Gidney, Craig. *"Algorithmic Assertions"* Google AI, http://algassert.com/. Accessed: December 12 2016

For extended reading list, consult source code, available at:
https://www.github.com/Macrofarad/ABriefIntroductionToQuantumComputingFromThePerspectiveOfLadderLogic