A Quantum Computing Arithmetic-Logic Unit

Ethan Butler, David Carroll, Bryson Phillip, Ben Ulrich

College of Computing and Software Engineering, Kennesaw State University

CS 4850: Senior Project

Professor Sharon Perry

December 9, 2022

Abstract

We show that a quantum version of a classical arithmetic-logic unit (ALU) can be implemented on a quantum circuit. It would perform the same functions as a classical ALU, with the possibility of adding quantum functions in conjunction. To create the quantum ALU, we utilized IBM's Python package Qiskit and JupyterLab. We believe that a quantum ALU has the potential to be faster than its classical counterpart and the ability to calculate quantum specific operations. The simple classical functions translated to a quantum circuit show a promising future for the development of a full quantum ALU with unique quantum operations.

Table of Contents

Introduction

The goal of our research was to demonstrate the expansion of the capabilities of a full adder into an Arithmetic-logic Unit (ALU), using quantum computing principles. A full adder in the classical sense, is one basic example of Boolean circuitry that is used to add binary numbers. As a unit, it can be used to compose many other complex circuits and can be strung together to add numbers of an indefinite size. Additionally, slight variations on its use can allow for subtraction, multiplication, and other more abstract operations. However, the uses for this unit of computation are not restricted to a classical computer. They may very well hold a similar status in future quantum algorithms. In this paper, we attempt to elucidate how the combinational logic of an adder can be emulated and expanded upon in a quantum computer. Our designs can be used as a reference or guide for future research. This research can expand upon our attempts to bridge quantum and classical computation.

Background

Our work was inspired by *Quantum Computing based Implementation of Full Adder* by Sohel et al. [6] which provided the inspiration and initial design components for our work. They showcased a full adder implemented with IBM Quantum Composer [1]. It demonstrates how quantum gates are analogous to classical gates with references to bra-ket notation, truth tables, and graphical results. It also serves as a good introduction to the visual layout of IBM Quantum Composer. We originally made the various operations of the ALU independently from each other. After that, we developed a way to use multiple operations inside of a single circuit. Next, we used IBM Quantum Composer [1] to create a visual layout of our circuit to solidify our understanding. This diagram can be seen in Figure 1. Afterwards, we coded the circuit itself in Qiskit assimilating and building off of our previous work.

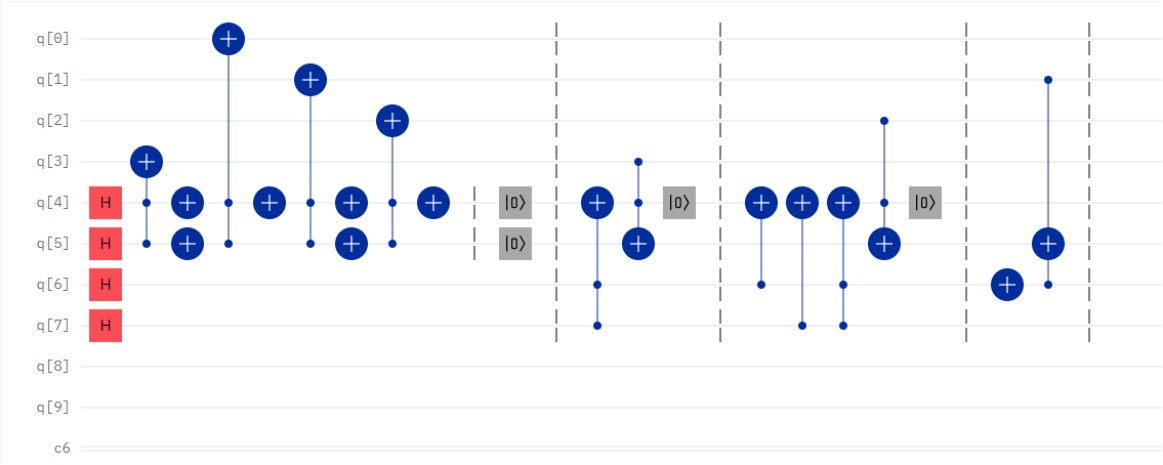

Fig. 1 The circuit in progress. This is an image of IBM's composer, which aided us in creating a visual for our quantum ALU

Software Development Life Cycle

Requirements

Over time we were able to distill our research into a set of requirements. They were to produce a formal research paper which would include an explanation of the classical and quantum circuitry involved in creating a numerical adder and other extended arithmetical operations. This paper should also showcase some of our code and the results of implementing these quantum circuits. A comparison and analysis of quantum and classical circuits would be shown. A GitHub repository needed to be shared which contained our code for the implementations of a full adder and ripple carry adder, in addition to the project's documentation.

Technology

For development on this project, we set up local environments using Anaconda to maintain consistency within our codebase. This allowed new developers to start development

quickly by running a set of commands to install all necessary project dependencies. This also created a virtual environment which isolated their project from the rest of their local system.

On our local machines we also used JupyterLab to help create and run our code. Inside of JupyterLab we used, jupyterlab-git, which is an extension for JupyterLab to help make integration with Git easier. This provided a GUI to complete Git functions such as staging, committing, and pushing within JupyterLab itself. Alternatively, a developer could use a terminal to work with Git if they preferred..

Dr. Dan Lo acted as our research advisor and project owner. He instructed us to use Python for our project, which is a standard language used for quantum computing research. We decided on using Python version 3.10, which at the initial time of the project, was the newest major release of Python. To format our code we used Black, a formatter for Python to ensure we did not have inconsistencies between development versions. We integrated this with a pre-commit Git hook that would check if our code didn't conform to Black's standards; if it didn't, the commit would be rejected and Black would automatically format the code.

Dr. Lo also told us to decide between Google's Cirq and IBM's Qiskit. They are both common Python packages used for quantum computing development. We decided to use Qiskit because there are more online resources for learning it compared to Cirq. To run the circuits themselves we utilized IBM's Quantum Lab with their API. This allowed us to run our quantum circuits on quantum computers and quantum simulators. We provided an individual account token in our code which authenticated us to use their API. In regards to circuit design, we occasionally used IBM Quantum Composer which allowed us to mock up our circuits before writing the code itself.

To facilitate the onboarding of developers getting started with our code, we created a website which contains documentation. It also contains general information regarding our project. It was built with the SSG (static site generator) Jekyll. We chose a theme and then

made most of our changes using Markdown, a markup language. Whenever a push is made to the *gh-pages* branch, a GitHub action will run and will update the website accordingly. This integration with GitHub and Jekyll is a benefit of using Jekyll as our SSG, since it is set up automatically.

We also are submitting a manuscript to the ACMSE 2023 conference. This requires us to make our paper using LaTeX, and a specific template they provide. Based on Dr. Lo's advice, we are using Overleaf to write our paper. It provides a LaTeX compiler, real time document editing by multiple people, and integration to a GitHub repository. We have an additional private GitHub repository dedicated to our paper, which provides a secure backup and version control.

Development

Our designs were also heavily influenced by the circuits shown in Sohel et. al's paper [6]. Our initial tasks focused on completing tutorials and informational guides on quantum computing topics. Some of the resources we used are the IBM Qiskit textbook, IBM's text on linear algebra for quantum computing and various other online sources that describe the nature of quantum computations [2]. We also placed a heavy emphasis on instilling version control early on. We created a Git repository to store all of our code, which included several branches for each team member's contributions. We also created a Discord server for communication. On this server, we communicated about collaboration, exchanged resources, posted deadlines/meeting information, and planned group meetings.

The beginning stages of our development involved doing a cursory survey of topics related to quantum computing. Some of us did research on such topics as: data structures in quantum computing, single event upsets, and linked lists. We then presented these findings in our weekly meetings to Dr. Lo. After this, we researched articles on contemporary developments and decided to focus on designing a quantum ripple carry adder as a continuation of a paper that documents a full adder circuit [6]. Our original design for the adder was based on the design in the paper with additions that allowed for passing carry bits from one place value to another. At this time, a major roadblock was that the circuit was too large to run on anything but a quantum simulator. This made the issue of doing time measurements and comparing the efficiency of classical and quantum circuits a problem. Also, even if we could solve this problem, finding comparable measurements for a classical circuit also proved to be ambiguous. We looked into solutions such as query complexity, doing time analysis using Python code, and performing simulations of a classical circuitry.

After stagnating on these issues, we continued on and developed general purpose code that could create a variable number of carry operations. However, upon the completion of this

code and talking with our project advisor, we decided to shift our development towards a quantum ALU. This new direction was still related to the future work that was proposed in the original paper. It also allowed for us to expand on several threads of future work within the scope of a single final product. Additionally, it gave us an opportunity for open-ended development, where we could utilize quantum effects to alter the nature and function of a traditional ALU beyond its normal capabilities.

Results

Our final ALU is capable of correctly computing AND, OR, NOT, and ADD operations for 2 qubits and an additional carry qubit for the addition operation. The architecture of our circuit is designed with minimal parallel computation and is mostly serial. This is done to conserve qubits, which are a more limiting factor than circuit depth when it comes to resources on an actual quantum computer. The depth of our circuit is thirty, indicating that at least thirty serial gate operations must be done back-to-back for any single qubit.

Operational Structure

The circuit is organized to compute outputs for all four operations and ignore the ones that aren't implied by the two-bit opcode. Three qubits are used for data input, two are used for the opcode, four are used for the opcode circuitry, and one is used for the output for a total of ten qubits in the ALU. In the figure below, there are two Hadamard gates on qubits four and five. These qubits represent the control bits, which dictate which operation is performed. The operations are mapped from qubits zero through three. Qubit zero maps to the ADD operation. Qubit one maps to the NOT operation. Qubit two maps to the OR operation, and qubit three maps to the AND operation. The Hadamard gate could produce one out of four outputs: (0,0), (0,1), (1,0), and (1,1). The output from the Hadamard gate is entangled with qubit three, which as stated before maps to the AND function. Then the fourth and fifth qubits (still the control bits) are flipped using the NOT gate. The results are entangled with qubit zero. Then there is a NOT gate applied to the fourth qubit, resulting in another possible operation, which is mapped to the first qubit. Then both the fourth and fifth qubit have a NOT gate applied to them to create the final possible operation, which is mapped to the second qubit. Further in the circuit, each possible single-bit operation will be computed. The output of that particular operation and its

corresponding indicator qubit (0 - 3) will be targets for the overall output. Basically, if the output of an operation is |1⟩, and the qubit indicator for that operation is also |1⟩ then that result will be the output of the ALU.

Qubits six and seven are the input bits. Shown below, there is a NOT gate over qubit six. Therefore the inputs are one and zero, respectively. Qubits eight and nine are used for the adder operation in the circuit. The eighth qubit is the carry-in bit and the ninth qubit is the carryout bit.

The first operation is the ADD. It starts with zeroing out the fourth and fifth qubits, which were formerly the control bits. Then two Toffoli gates (also known as double controlled not gates) are applied to create the desired result, which is mapped to qubit three.

The second operation is the OR. The fifth qubit is zeroed out once again. Then two controlled NOT gates are applied along with two more Toffoli gates to produce the desired result which is mapped to qubit two.

The third operation is the NOT. The fifth qubit is zeroed out and then a NOT gate is applied to it. A controlled NOT and Toffoli gate are then applied to create the desired result which is mapped to qubit one.

The fourth and final operation is the ADD. The fifth qubit is zeroed out again. Then three controlled not gates are applied along with two NOT gates and three Toffoli gates. The result is mapped to qubit zero, and the carry-out bit is mapped to qubit nine.

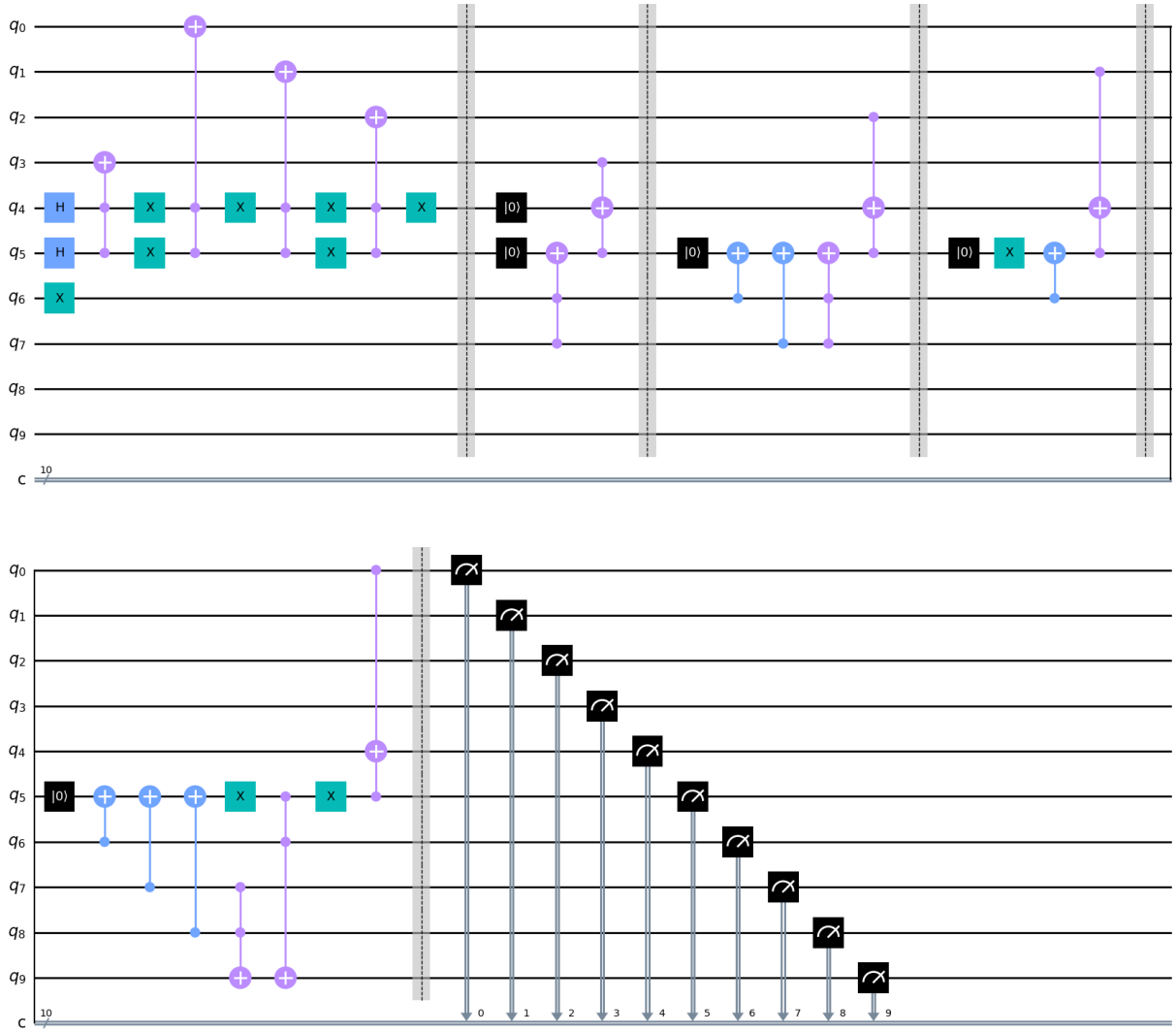

Fig. 2 The full quantum Arithmetic-logic Unit; image created with IBM's Qiskit Virtualization module which is part of the Qiskit Python package. Qubits one through three are the data input, four & five are the opcode qubits, and six through nine are for recording the output.

Every bit is measured at the end of the circuit, which produces the histogram seen below. We ran the circuit one hundred times on the simulator, which created the results. The zeroes and ones beneath the bars represent qubits zero through nine. Qubit zero is on the far right and qubit nine is on the far left. The number above the bar simply represents the probability that the exact set of zeros and ones appears when the circuit runs. This would be

correct, as the Hadamard gates create four different possible scenarios. As the number of shots increases, the higher the accuracy of the probabilities.

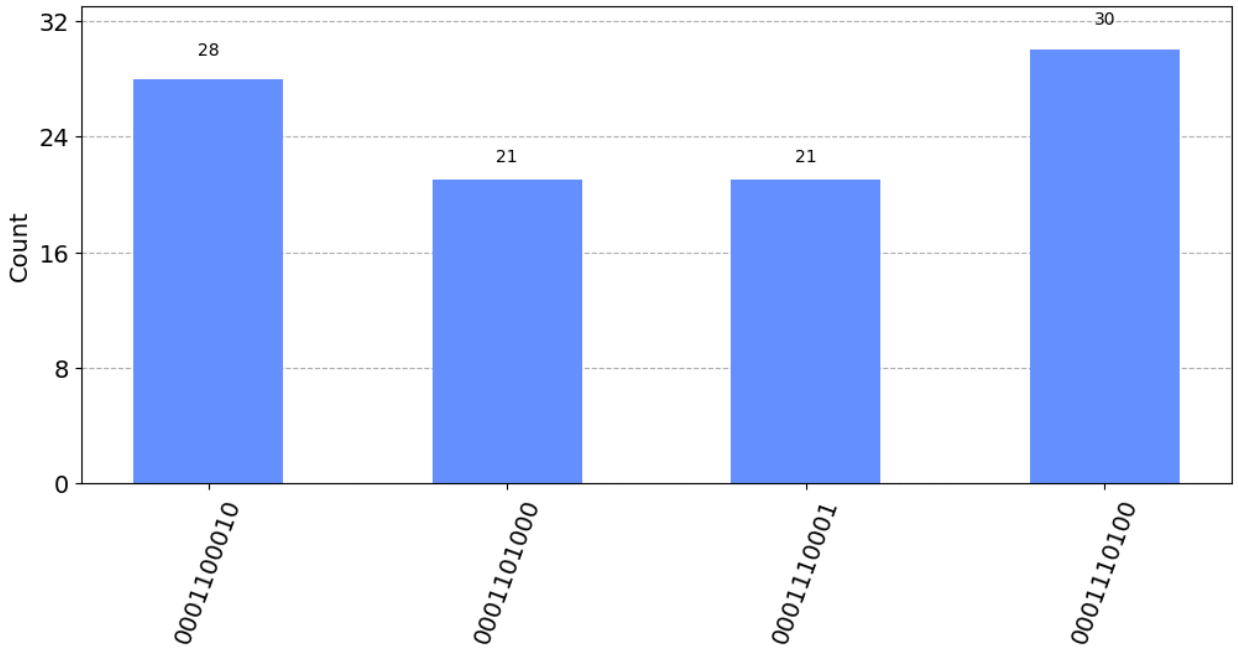

Fig. 3 Histogram showing probabilities for operations. Qubits are read right to left (i.e. the last three qubits are the input).

Known Complications

Our team has encountered some constraints on the extent of testing that can be done on many of our circuits. As it stands when using the proprietary IBM Qiskit [8] framework, all free quantum back ends are capped at seven qubits. The final design for our ALU is ten qubits. The issue of doing circuit analysis and complexity analysis is ambiguous at times, even for classical

algorithms running on sequential logic. This difficulty is further increased when dealing with circuitry instead of programs, and when dealing with quantum mechanical effects. The imperfect nature of quantum states means that they can be interfered with in a number of ways that are not always deterministic. This also means that the time needed for operations can vary. However, these differences are largely dependent on the quantum back-end itself. Noise is another issue in quantum circuitry. There are many models for noise and methods for correcting for noise that are outside of the scope of our research. Redundancy of computation is a basic way to reduce noise at an increase in computational time [7]. There is also a trade off between time and space. Doing redundant calculations in our circuitry would increase the circuit depth and/or its qubit count, but may decrease the number of shots needed to discern the mode of the results.

Project Planning and Management

Collaboration

Our primary form of communication within our group was through the app Discord. This allowed us to communicate via text and voice chat. It also allowed us to organize information about the project into different channels. For example, we had a *resources* channel which contained resources for quantum computing, Qiskit, Git, and other relevant material to the project. The other channels were *general*, *meeting-notes*, *meeting-log*, *documents, deadlines-and-schedule*, *research-papers*, *gantt-chart-time*, *research-report-resources*, and *c-day*.

We met most Tuesdays and Thursdays to discuss our project in person from 6:30 PM to 7:45 PM if class attendance was not required. In addition to this, we had weekly meetings on Microsoft Teams with Dr. Lo and the other quantum computing senior project group on Thursdays from 3:30pm to approximately 4:15pm. This allowed us to video chat while also being able to share our screen to showcase our progress. Occasionally, if a meeting was canceled, we would email our progress to Dr. Lo.

To work on our code with each other, we utilized GitHub with Git. Further details about this are shown in the version control section. For working on our documents, we used Google Docs, Google Slides, and Microsoft PowerPoint which allowed for real time editing by multiple people. In addition to this, we used Overleaf to edit our LaTeX documents for our formal research report.

Gantt Chart

Project Name: DL-01, Quantum Computing, 4850-01 Sr. Project, Fall 2022, Sharon Perry
Report Date: 11/28/2022

					Milestone #1				*Milestone #2*				C-Day	
Deliverable	**Tasks**	**Complete%**	**Current Status Memo**	**Assigned To**	**09/27**	**10/04**	**10/11**	**10/18**	**10/25**	**11/01**	**11/08**	**11/15**	**11/17**	**11/24**
Research Report	Resource Collection	100%	Completed	Ben, Bryson, David, Ethan	1		1							
	Background Research	100%	Completed	Ben, Bryson, David, Ethan	15	24	12	15		6	1		2	
	Write Reports	60%	Much of the content is completed but need to convert it to LaTeX	Ben, Bryson, David, Ethan					12	1	5	1	1	1
Development	Qiskit Basics	100%	Completed	Ben, Bryson, David, Ethan	1		1			1				
	GitHub Setup	100%	Completed	Ben, Ethan		3		2	2	2				
	Implement a quantum full adder	100%	Completed	Ben, Bryson, David, Ethan	3	3	1							
	Implement a quantum ripple-carry adder	100%	Completed	Ben, Bryson, David, Ethan				3						
	Documentation setup	80%	Most documentation is complete, but need to finish the website.	Ben, Ethan				2			3			
	Implement additional operations	100%	Completed	Ben, Bryson, David, Ethan				3	9	14				
Final report	Poster preparation	100%	Completed	Bryson, David								3	9	
	Presentation preparation	75%	Prepared all materials but just need to practice our speeches.	Ben, Bryson, David, Ethan									4	
	Final report submission	80%	Most of it is completed but need to submit by the final deadlines	Ben, Bryson, David, Ethan							2		2	20
			Total work hours	**191**	**20**	**30**	**15**	**25**	**23**	**24**	**11**	**4**	**18**	**21**

Legend
Planned 335
Delayed
Number Work: man hours

Fig. 4 Gantt Chart. This is an image of our Gantt chart that we used to track hours worked on tasks by individuals in the group.

Version Control

In order to manage version control between all members of the project, we decided to utilize the GitHub platform to host our repository and then manage access to it through the use of Git, specifically the open source add-on to JupyterLab [10]. To manage commits between versions, we used the Python formatter, Black [11], as part of a pre-commit Git hook to format our code for the least differential code. We also included a .gitignore file that would exclude the “ibmqID.py” file, which contained each members’ unique IBMQ identifier and therefore should be different for each member, as well as checkpoint files, which were temporary files only existing during runtime.

To collaborate efforts between members, we had to come up with a version control workflow so that we did not impede each other's progress. Each member was responsible for their own branch, only pushing to a staging branch when a task was completed. We would then conduct a code review to ensure our combined program functioned correctly before finally pushing our updated final product to our main branch.

Test Plan and Test Report

Test Plan

Testing was to be done on the results of the program as well as its efficiency. When run on a quantum or classical circuit simulator, we would look for accuracy of results to determine if bugs were present, comparing our results to those of a given set of test cases. For testing on quantum computers, we would analyze the time and memory required to complete jobs as well as the noise encountered in our results. We would compare testing done on quantum computers to the simulated tests and use this as a basis for our research.

Test Report

In regards to the test plan, we realized that doing analysis of our quantum circuits would be outside the scope of our project, so we ended up deciding to focus our research on the implementations of the circuits themselves. We added multiple operations to our original adder as planned, but also devised a way to integrate the operations together into one circuit. This provides a base for a quantum ALU which acts as a “black box”.

Conclusion

Our paper demonstrates a quantum computing Arithmetic-logic Unit with various operations. As classical ALUs have many operations, people should be able to use our work to implement more operations than the ones we showcase. As the technology around quantum computing grows, the implementation of quantum ALUs may become more common. The demand for higher computational speeds is growing exponentially. Classical hardware is so limited compared to quantum [5], giving quantum a promising future. Through our research, we look to the near future where people can build off our work to make quantum computing more ubiquitous.

Future Research

Quantum computing has a significant amount of progress that can be made. There are multiple things that can be done to expand upon our initial work. Some examples are adding more classically analogous operations. This would make it more comparable to a standard classical ALU. In addition, implementing quantum operations specifically inside the ALU would be a novel approach that could possibly improve the efficiency to complete quantum computations. The continuous nature of qubits allows them to store more information than a classical bit. However, this information tends to be transient and difficult to retrieve [9]. Methods of reducing the ambiguity that is inherent in a qubit's quantum state will allow for the possible encoding of more information per bit. Altering the quantum phase of a qubit using reversible operations and recording its value into other qubits could yield more information than simple measurement and may open up more possibilities. These possibilities may even expand the range of functionality for our quantum ALU design. Alternatively, research could be directed towards the feasibility of an ALU in bridging the gap between the computational domain of classical and quantum computers. Quantum computers, while theoretically impressive, are mainly useful in optimization problems and do not handle general purpose computing all too well [5]. Our models could break ground in the direction of making more accessible problems available to these machines. More testing could be done on using an ALU for complicated, multi-step problems. Our circuit was designed to operate in a superposition until the end of a very small computation. An alteration of our design would allow for multiple steps to occur before decoherence. Specifically, data on the decline in output accuracy could be important in determining if additional error correction techniques are needed [4]. These alterations may involve changes to our virtual circuit design and other related designs.

Acknowledgements

Special thanks to Dr. Dan Lo who acted as our research advisor and gave instrumental guidance for completing our work and learning about quantum computing.

Thank you to Professor Sharon Perry who acted as a project advisor, provided external resources for learning about quantum computing, and gave us advice on the direction of our work.

We acknowledge the use of IBM Quantum services for this work. The views expressed are those of the authors, and do not reflect the official policy or position of IBM or the IBM Quantum team.

References

[1] IBM [n. d.]. *IBM Quantum Composer*. IBM. https://quantum-computing.ibm.com/composer/

[2] IBM 2022. *Qiskit 0.39.3 documentation*. IBM. https://qiskit.org/documentation/

[3] IBM 2022. *Qiskit Terra*. IBM. https://github.com/Qiskit/qiskit-terra

[4] Daniel Azses, Maxime Dupont, Bram Evert, Matthew J. Reagor, and Emanuele G. Dalla Torre. 2022. Navigating the noise-depth tradeoff in adiabatic quantum circuits. https://doi.org/10.48550/ARXIV.2209.11245

[5] Nilesh Barde, Deepak Thakur, Pranav Bardapurkar, and Sanjaykumar Dalvi. 2011. Consequences and Limitations of Conventional Computers and their Solutions through Quantum Computers. *Leonardo Electronic Journal of Practices and Technologies* 10 (07 2011), 161–171.

[6] Mohammed Arifuddin Sohel, Naailah Zia, Mohd Akef Ali, and Nida Zia. 2020. Quantum Computing based Implementation of Full Adder. In *2020 IEEE International Conference for Innovation in Technology (INOCON)*. 1–4. https://doi.org/10.1109/INOCON50539.2020.9298394

[7] Andrew M. Steane. 1996. *Quantum Error Correction*. The Royal Society. https://users.physics.ox.ac.uk/~Steane/qec/moriand.html

[8] https://github.com/Qiskit/qiskit/blob/master/Qiskit.bib 2021. Qiskit: An Open-source Framework for Quantum Computing. https://doi.org/10.5281/zenodo.2573505

[9] John Wright. 2015. Lecture 18: Quantum Information Theory and Holevo's Bound. https://www.cs.cmu.edu/~odonnell/quantum15/lecture18.pdf. Accessed: 2022-11- 30.

[10] Jupyterlab. "A Git Extension for JupyterLab." *GitHub*, https://github.com/jupyterlab/jupyterlab-git.

[11] Psf. "The Uncompromising Python Code Formatter." *GitHub*, https://github.com/psf/black.

Appendix A: Glossary

ALU - Arithmetic-logic Unit. Performs the arithmetic and logic operations of a computer.

Controlled NOT gate - A form of entanglement where the quantum state of a target qubit is flipped if a control qubit is in quantum state of $|1\rangle$.

Entangled bits - When the measurement outcome of one qubit is statistically related to the measurement outcome of another.

Hadamard Gate - A gate that puts a qubit into a mixed state (equal probability of measuring $|1\rangle$ and $|0\rangle$).

Noise - Deviation from expected results when a circuit is run on a quantum computer. Results can come from different sources such as cell phones, the earth's magnetic field, other qubits, etc.

Qubit - A quantum bit. It is analogous to a bit in classical computing.

Shots - The number of times the simulator will run the circuit.

Toffoli Gate - A C-NOT gate with two controls, where both should be $|1\rangle$ to flip the target.

Bra-Ket Notation - Bra-Ket notation is an altered vector notation for quantum mechanical superpositions. A superposition can be stored as a unit vector with a basis of $|1\rangle$ and $|0\rangle$. $|1\rangle$ represents a superposition where the probability of measuring 1 is 100%. $|0\rangle$ represents a 0% probability of measuring 1. Coefficients attached to these components are used to represent a composite vector. The $|\ \rangle$ is used to represent a ket which is a column vector. $\langle\ |$ is used to represent a bra which is a row vector. When representing a superposition in both bra and ket form, the two vector expansions are conjugates of each other. This allows the complex component to be eliminated when taking the dot product.

Appendix B: Online Quantum Computing Resources

This section consists of a handful of resources that were helpful to us to learn quantum computing and Qiskit. We hope that you will find these just as elucidating as we did. Many thanks to those who made these available to the public.

Free Resources

• IBM Quantum Lab User Guide: https://quantum-computing.ibm.com/lab/docs/iql/

• Learn Quantum Computation Using Qiskit: https://qiskit.org/textbook/

• Qiskit Documentation: https://qiskit.org/documentation/

• Qiskit Textbook (beta): https://qiskit.org/learn/

• Quantum Computing For The Very Curious: https://quantum.country/qcvc

Paid Resources

• Practical Quantum Computing: Asynchronous Workshop: https://ksu-quantum-capstone.github.io/CS4850-DL1/ws/

Note: we are not affiliated at all with this workshop, but it was helpful for us in learning more about quantum computing. However, we were able to receive a discounted group rate.