A Quantum Computing Arithmetic-logic Unit

Bryson Phillip
Kennesaw State University
Kennesaw, Georgia, USA
biology516@gmail.com

Ethan Butler
Kennesaw State University
Kennesaw, Georgia, USA
ethan@ethanbtlr.com

Ben Ulrich
Kennesaw State University
Kennesaw, Georgia, USA
bg.ulrich@gmail.com

David Carroll
Kennesaw State University
Kennesaw, Georgia, USA
dmcarroll19@gmail.com

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.

CCS CONCEPTS

• **Hardware** → *Quantum computation*; *Combinational circuits*; • **Computer systems organization** → **Quantum computing**.

KEYWORDS

Quantum Computing, Qiskit, Arithmetic-logic Unit, Qubit

ACM Reference Format:
Bryson Phillip, Ethan Butler, Ben Ulrich, and David Carroll. 2023. A Quantum Computing Arithmetic-logic Unit. In *2023 ACM Southeast Conference (ACMSE 2023), April 12–14, 2023, Virtual Event, USA.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3564746.3587005

1 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.

ACMSE 2023, April 12–14, 2023, Virtual Event, USA

ACM ISBN 978-1-4503-9921-0/23/04…$15.00
https://doi.org/10.1145/3564746.3587005

2 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 [3]. 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 [3] 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.

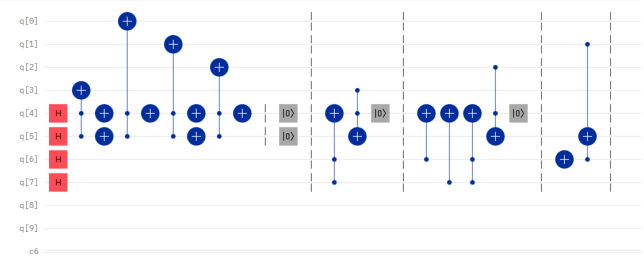

Figure 1: Circuit Made Using IBM Quantum Composer

3 A QUANTUM-COMPUTING ARITHMETIC-LOGIC UNIT

Our ALU is capable of correctly computing AND, OR, NOT, and ADD operations for two 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 30, indicating that at least 30 serial gate operations must be done back-to-back for any single qubit.

3.1 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 10 qubits in the ALU.

In Figure 2, there are two Hadamard gates on qubits 4 and 5. These qubits represent the control bits, which dictate which operation is performed. The operations are mapped from qubits 0 through 3. Qubit 0 maps to the ADD operation. Qubit 1 maps to the NOT operation. Qubit 2 maps to the OR operation, and qubit 3 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 3, which as stated before maps to the AND function. Then the 4th and 5th qubits (still the control bits) are flipped using the NOT gate. The results are entangled with qubit 0. Then there is a NOT gate applied to the 4th qubit, resulting in another possible operation, which is mapped to the 1st qubit. Then both the 4th and 5th qubit have a NOT gate applied to them to create the final possible operation, which is mapped to the 2nd 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 $|1\rangle$, and the qubit indicator for that operation is also $|1\rangle$ then that result will be the output of the ALU.

Qubits 6 and 7 are the input bits. Shown in Figure 2, there is a NOT gate over qubit 6. Therefore the inputs are 1 and 0, respectively. Qubits 8 and 9 are used for the adder operation in the circuit. The 8th qubit is the carry-in bit and the 9th qubit is the carryout bit. The first operation is the ADD. It starts with zeroing out the 4th and 5th 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 3. The second operation is the OR. The 5th 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 2. The third operation is the NOT. The 5th 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 1. The fourth and final operation is the ADD. The 5th 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 0, and the carry-out bit is mapped to qubit 9.

All figures were created with IBM's Qiskit Virtualization module, which is part of the Qiskit core Terra [4][5]. As can be seen in the figures, qubits 1 through 3 are the data input, 4 and 5 are the opcode qubits, and 6 through 9 are for recording the output. Every bit is measured at the end of the circuit, which produces the histogram seen in Figure 3. We ran the circuit 100 times on the simulator, which created the results. The zeroes and ones beneath the bars

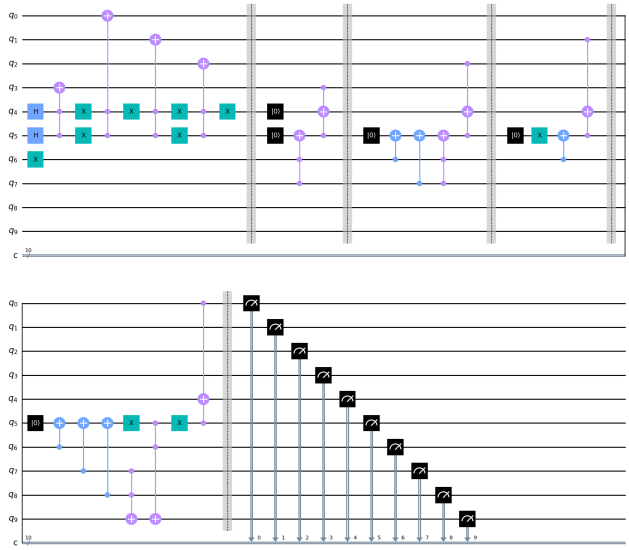

Figure 2: The Quantum Arithmetic-logic Unit

represent qubits 0 through 9. Qubit 0 is on the far right and qubit 9 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.

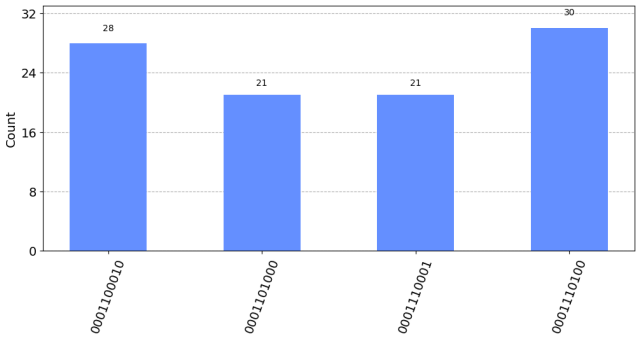

Figure 3: Histogram of Operations' Probabilities

3.2 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 10 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.

4 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 [2], 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.

5 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 [2]. 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 [1]. These alterations may involve changes to our virtual circuit design and other related designs.

ACKNOWLEDGMENTS

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] 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

[2] 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.

[3] IBM Quantum Composer. 2022. *IBM Quantum Composer.* IBM. https://quantum-computing.ibm.com/composer/

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

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

[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).* IEEE, Bangluru, India, 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] Matthew Treinish, Jay Gambetta, Paul Nation, Paul Kassebaum, Diego M. Rodríguez, et al. 2022. *Qiskit/qiskit: Qiskit 0.39.3.* IBM. https://doi.org/10.5281/zenodo.7363289

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

A GLOSSARY

- Arithmetic-logic Unit (ALU) - 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.
- Mixed State - There is an equal probability of measuring $|1\rangle$ and $|0\rangle$.
- Noise - Deviation from expected results when a circuit is run on a quantum computer. Resulting noise 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 quantum 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.

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.

B.1 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

B.2 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.

C RUNNING THE CIRCUIT

C.1 Getting Started with Our Code

(1) Install anaconda from: https://www.anaconda.com/.
(2) Once installed, in your Anaconda Prompt on Windows or terminal on Mac/Linux, run the command
`conda list`
to test your installation.
If it's installed correctly, a list of installed packages will appear. If you run into trouble visit: https://docs.conda.io/projects/conda/en/latest/user-guide/install/.
(3) Run the command
`conda create --name quantum python=3.10`
to create a new conda environment call "quantum" with a Python 3.10 installation.
(4) Use the command
`conda activate quantum`
to activate the virtual environment.
(5) Install JupyterLab
`conda install -c conda-forge jupyterlab`
(6) Install JupyterLab's git extension
`conda install -c conda-forge jupyterlab-git`
(7) Install pre-commit and Black
`conda install -c conda-forge pre-commit black`
(8) Install Qiskit
`pip install qiskit`
(9) Install visualization support for Qiskit
`pip install qiskit[visualization]`
If this does not work it may be because you are using zsh, which is the default shell on newer MacOS systems. Use this command instead:
`pip install 'qiskit[visualization]'`
Take notice of the extra quotes.
(10) To deactivate your current conda environment use the command
`conda deactivate`.

Feel free to explore our work at https://ksu-quantum-capstone.github.io/CS4850-DL1/.

C.2 System Information

See Table 1 for information about the system components used to run the circuit. See Table 2 for version information of the key software components used with our code-base.

Table 1: General System Information

Information	Value
Python compiler	GCC 11.2.0
Python build	main, Nov 24 2022 14:13:03
OS	Linux
CPUs	4
Memory (Gb)	15.586982727050781
Circuit's run date and time	Wed Nov 30 17:18:31 2022 EST

Table 2: Software Versions

Software	Version
qiskit-terra	0.22.3
qiskit-aer	0.11.1
qiskit-ibmq-provider	0.19.2
qiskit	0.39.3
jupyterlab	3.5.0
jupyterlab-git	0.40.1
pre-commit	2.20.0
black	22.10.0
conda	22.9.0
conda-build-version	3.23.1
python version	3.10.8