# Efficient Quantum Multiplication in the Quantum Fourier Transform Domain

Aden Crimmins
abc8255@rit.edu

# Efficient Quantum Multiplication in the Quantum Fourier Transform Domain

Aden Crimmins

# Efficient Quantum Multiplication in the Quantum Fourier Transform Domain

Aden Crimmins

May 2024

A Thesis Submitted
in Partial Fulfillment
of the Requirements for the Degree of
Master of Science
in
Computer Engineering

RIT | Kate Gleason College of Engineering

*Department of Computer Engineering*

# Efficient Quantum Multiplication in the Quantum Fourier Transform Domain

Aden Crimmins

**Committee Approval:**

| | |
|---|---|
| Sonia Lopez Alarcon *Advisor* | Date |
| Department of Computer Engineering | |

| | |
|---|---|
| Michael Zuzak | Date |
| Department of Computer Engineering | |

| | |
|---|---|
| Marcin Lukowiak | Date |
| Department of Computer Engineering | |

## Acknowledgments

I'd like to thank all my research colleagues, past and present, for the support that was given as I progressed through this thesis. From tips and tricks to full discussions, they were pivotal in my understanding of the subjects covered, and I couldn't have made it this far without them. Finally, I would like to thank my advisor, Dr. Sonia Lopez Alarcon, for supporting my research as well as keeping me on the right track with clear targets and guidance through the whole process.

*This work is dedicated to my family and my fiancée Savannah*

# Abstract

The subject of Quantum Computing is in its early stages of development, and new technologies and algorithms are constantly evolving. Even with all of this progress it is important to recognize that current quantum computers are still in their infancy and, as such, have notable issues. The “era” that we are in currently is known as the Noisy Intermediate-Scale Quantum (NISQ) era, which is characterized by machines of small size that are prone to quantum noise, which causes computational error the longer the circuit runs. In order to make use of these quantum computers, algorithms need to be developed that both use a minimal number of qubits and minimize the time needed for them to run. Utilizing classical computing techniques as the inspiration for this work, the aim is to reduce ”depth,” which is analogous to the run-time of a quantum circuit. By taking advantage of the parallel computing techniques in classical arithmetic logic units (ALUs), we are able to create quantum circuits that can take advantage of their own parallelism, leading to a reduced run-time.

In addition to the potential benefits to the complexity of the algorithm, decreasing run time reduces the potential for error to accrue and compound as the circuit runs. This approach was chosen because, while other approaches exist which focus on reducing the effects of noise, this one aims to increase the speed of the circuit itself, leading to designs that will still be preferable on future machines. Much like classical computing, small-scale circuit designs are used as building blocks for large-scale algorithms. Improvements made to a quantum multiplication algorithm could prove useful for different applications, such as modular multiplication in Shor’s algorithm [1], or could play a role in the implementation of oracles for Grover’s search algorithm [2]. The designs presented in this thesis offer an efficiency improvement over existing quantum multipliers coming from three sources: the efficient use of addition on the phase domain (a quantum property), the classical hardware design of array multipliers, and the use of approximation in the phase domain.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

## 1.1 Motivation

Quantum computing is still in its early stages of development, but it has the potential to revolutionize many fields, including pharmacology, materials science, and artificial intelligence. In order for quantum computations and quantum computers to have a meaningful impact on industry and the world at large, they must be capable of providing a Quantum Advantage (QA). The definition of QA is the demonstrable and measurable success of processing some real-world problem faster or more efficiently on a quantum computer than on a classical computer. Much of the research in this field focuses on identifying algorithms which provide speedups compared to those that are possible on a classical computer. One commonly referenced example is Shor's algorithm [3], which can factor large numbers exponentially faster than the best-known classical algorithms. In addition to Shor's algorithm, there are other notable algorithms that could provide a speedup compared to classical alternatives if implemented properly on a physical quantum computer. Some examples include Grover's Algorithm for unsorted search [2], Quantum Unconstrained Binary Optimization, Variational Quantum Eigensolvers, and more.

In the Noisy Intermediate-Scale Quantum (NISQ) era [4], noise is a critically limiting factor. In addition, computations are constantly challenged by the decoherence time, the time that takes for a quantum system degrade and fall out of its quantum state. The circuits that describe quantum computations are characterized by the number of qubits that they require or *width* and the number of computational steps necessary to completion or *depth*. In the NISQ era, circuits must reach completion before qubits fall out of their delicate quantum state. In order to counter the noise inherently present in these systems, and to make them more efficient, a focus must be placed on reducing the time that each algorithm is run. As technology improves and the number of qubits available grow while the error and noise levels decrease, reaching post-NISQ era, the depth of the circuit will still be a critical factor that needs to be reduced, and with direct impact on the performance of the computation.

There are two notable issues with the use of these algorithms. Firstly, there is transfer to and from the quantum domain, which adds both complexity and time constraints that would not exist if the entire algorithm were confined to one of the domains [5]. Secondly, even if the computation was performed almost entirely in a quantum system, there is still the presence of noise, which causes errors that compound the longer the system runs. One method to reduce both of these issues is to create algorithms that provide functionality that may not be faster than classical systems but can be run in a quantum environment. This allows for fewer instances of boundary crossing and, if optimized properly, could be roughly equivalent to their classical counterpart.

The scalability problem of quantum circuit design is ubiquitous in this field, but it is exacerbated by naive design practices. Good understanding of the properties of quantum algorithms leads to more efficient designs, with reduced requirements for number of qubits and quantum gates. On the other hand, the description of quantum applications and circuits display strong similarities with the hardware design approaches that have been commonly used in classical computing, for example in the context of Field Programmable Gate Arrays (FPGAs), or Application Specific

Integrated Circuits (ASICs). Both are potentially described at the (qu)bit-wise level, through logical one and two-(qu)bit gates in a sort of Quantum Description Language —in parallel to Hardware Description Languages—. This paper looks into well known HDL approaches for inspiration, and proposes a new and efficient quantum integer multiplier design.

The focus of this work is on one such algorithm, which, in many of the different quantum implementations, is lagging behind its classical counterpart. From basic implementations of multiplication using repeated addition[6][7], to an algorithm utilized by IBM Quantum for integer multiplication[8], we compare different techniques to the one proposed by this work.

## 1.2 Objectives

In this thesis we propose a new quantum multiplication algorithm in the form of the Quantum Array Multiplier (QAM) along with the Approximate Quantum Array Multiplier (AQAM). The primary objective of this research is to outline, improve, and analyze a quantum multiplication algorithm that takes advantage of both the Quantum Fourier Transform (QFT) and the design of the classical array multiplier. A generalized algorithm will be developed, which will then be scaled to different degrees to get an estimate of the complexity of the algorithm in comparison to other existing algorithms. Additionally, further research into the QFT will be incorporated in the form of the Approximate Quantum Fourier Transform (AQFT), which, given large enough circuits, will reduce the complexity by limiting certain operations beyond a specific threshold.

Ultimately, the algorithm created through this research can help provide a backbone for current higher-level quantum applications. Since this design will be generalizable and not require any prior knowledge of the inputs it can be used as a insertable component, just as multiplication is one component of many different classical algorithms. This feature is not entirely present in the other algorithms being compared, either due to mid-circuit measurements required or identical input width. Additionally, further improvements and research could potentially help create an algorithm that is just as efficient, if not better than, the current best classical algorithms.

## 1.3 Thesis Contribution

Algorithms for performing arithmetic operations are an integral component for most computable operations. As improvements to these algorithms surface, the have beneficial rippling effects into other applications that utilize these algorithms. This thesis specifically focused on the operation of multiplication as an area available for improvement. The contributions of this thesis to the area of quantum arithmetic is:

- **A new quantum multiplication algorithm that significantly reduced the depth overhead for quantum multiplication, especially as input size increases.**

- **Provided a basis for the use of approximate phase estimation in quantum phase domain, while also showing the potential for more accurate results while on noisy hardware.**

- **Highlighted the scaling and accuracy of different multiplication algorithms with input size.**

# Chapter 2

# Background

## 2.1 Classical Multiplication

The design proposed in this work is based on the array multiplier structure, which is commonly used in classical ALUs to perform multiplication. The Array Multiplier is a combinational circuit that, in the classical domain, multiplies two binary numbers using a shifted array structure of Full Adder (FA)s as shown in Fig. 2.1. Similar to how multiplication is performed on paper, it finds the partial products and adds them together. The benefits of the array multiplier come in the form of it's repetitive shape and simplicity of design. This allows it to be easily abstracted in the creation of a generic algorithm that mimics the classical design. In the classical implementation there is a relatively high delay present that scales with the width of the input size for the multiplication. This is due to the fact that there is dependency on the outputs of the previous partial products. This limitation is touched on in 3.3, in which it is shown that in the quantum domain the delay can be reduced with the parallelization of operations performed.

In the classical design, each FA has three inputs as well as a sum and a carry output, which propagate through the circuit, eventually resulting in the product of the two inputs. In order for a FA to compute its two outputs, it relies on the carry from the previous column and the output of the FA in the previous row.

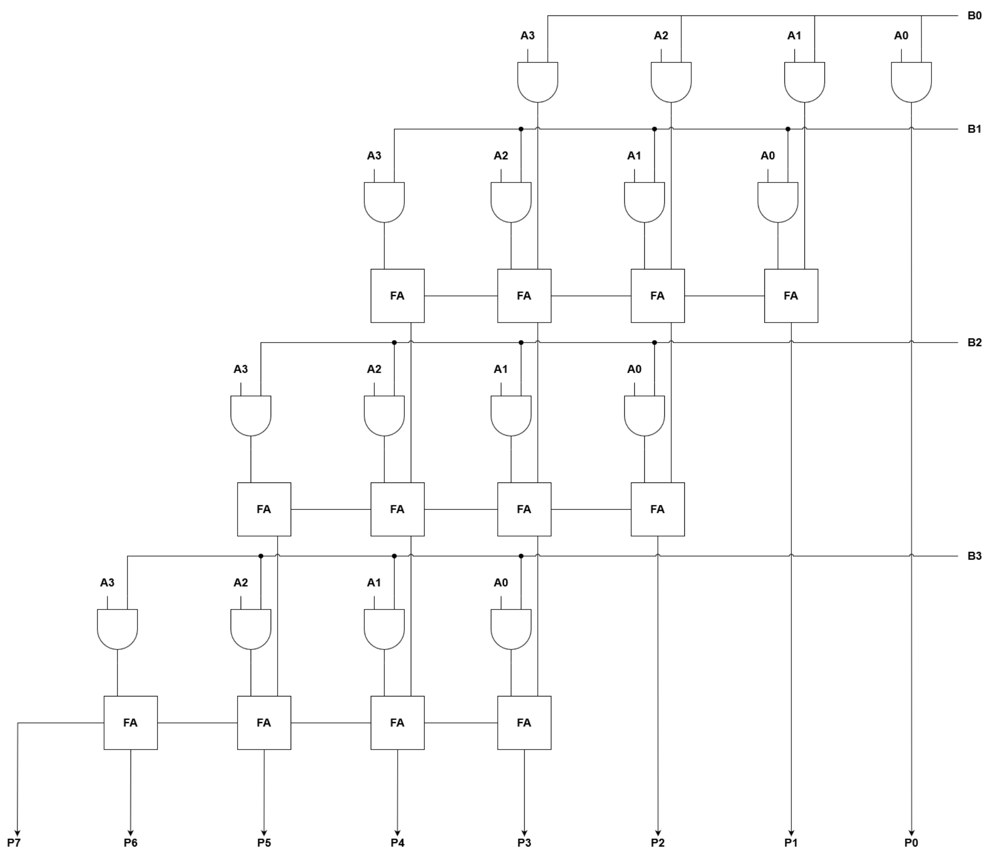


**Figure 2.1:** Visualization of an Array Multiplier architecture utilizing both full adders and half adders.

Due to the purely logic-based nature of the array multiplier, it is possible to directly implement the design using classically-analagous gates acting on qubits, but this method requires an excessive amount of resources. This complexity increases very quickly when considering the architecture of the array multiplier, which strings together n-1 rows of these ripple carry adders in order to perform $n * n$ multiplication, where n is the bit width of the inputs. A method of performing this multiplication in the phase domain using the QFT will be designed and compared to existing quantum multiplication algorithms to determine what relative speedup is achieved through such an algorithm.

## 2.2 Quantum Computing

Quantum computing is a field in which advanced principles of physics are utilized to perform computations that are otherwise impossible on our current classical computers. This subject started to grow in the latter half of the 20th century, though, it was relegated mostly to theory until the early 2000's as physical quantum computers started to come into existence[9][10]. These computers started very basic, but as time went on both their capacity and accuracy have been improving. Unlike classical computers, which represent data using bits of either 0 or 1, quantum computers use quantum bits (qubits), which can exist in multiple states simultaneously in a phenomenon known as superposition. In addition to superposition, some of the major concepts utilized in this work include entanglement as well as the QFT.

By utilizing these different properties available to quantum computers, one is able to achieve QA. While hardware has yet to scale to a point that QA can be shown for practical problems, it has been shown in sample problems and algorithms [11] [12]. The impact of such an advantage is projected for fields such as cryptography, machine learning, and even pharmaceutical drug development[13]. When manipulated correctly, the fundamental quantum characteristics can enable quantum algorithms to gain a QA over classical computers.

One of the most common frameworks for quantum computing is the quantum gate model. This model is built on the basis of qubits being acted upon by sets of unitary operators represented by "gates". Quantum gates act upon the qubits to alter their states, the evolution of which is governed by the operation of linear algebra in the Hilbert space. These gates come mainly in the form of 1 and 2-qubit gates as shown in Fig. 2.2.

The quantum algorithms are generated by sequentially applying these quantum gates to the qubits until they reach the desired state. Once this point is reached, a

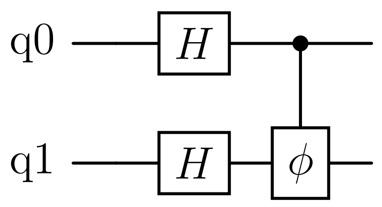


**Figure 2.2:** Example circuit which shows two single-qubit (Hadamard) gates followed by a two-qubit controlled phase shift gate.

measurement is taken, collapsing the superposition and returning the output of the algorithm. This formulaic approach to building algorithms through the combination of smaller components is a widely adopted method and will be utilized going forward in this research.

### 2.2.1 Superposition and Entanglement

In order to understand how qubits function, an basic understanding of superposition must first be gained. The idea behind superposition is that the state of a given qubit $|\phi\rangle$ can be expanded as a linear combination of eigenstates. These eigenstates form a complete basis and as such, can be combined to describe any given state. The two-dimensional basis states often associated with the classical values of 0 and 1 are $|0\rangle$ and $|1\rangle$, which have vector representations shown in equation 2.1.

$$|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \qquad |1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \tag{2.1}$$

When working with states that are a combination of multiple basis states it is important to note that the probabilities of each state must add up to 1. This is due to the fact that when measuring a qubit, you break down any superposition it may be in to yield either a $|0\rangle$ or a $|1\rangle$. For example given a perfect superposition between the 0 and 1 state you will have an equally likely, 50%, chance of measuring a 0 or a 1. This relationship can be represented more generally with equation 2.2 for a 1-qubit example where $|\alpha|^2$ is the percent chance of measuring $|0\rangle$ and $|\beta|^2$ is the percent chance of measuring $|1\rangle$. These variables $(\alpha, \beta)$ are complex numbers such that $|\alpha|^2 + |\beta|^2 = 1$.

$$|\phi\rangle = \alpha \begin{bmatrix} 1 \\ 0 \end{bmatrix} + \beta \begin{bmatrix} 0 \\ 1 \end{bmatrix} \tag{2.2}$$

As more qubits are connected, the dimension of the available state space grows exponentially, with the number of these basis vectors being equivalent to $2^n$, where $n$ is the number of qubits. When the state of this system cannot be expressed as the tensor product of the individual qubit states it is called an *entangled* state. The existence of these entangled states allows quantum computers an edge over classical computers making it possible to represent a complex $2^n$ dimensional vector space using $n$ qubits.

### 2.2.2 Quantum Fourier Transform

Taking root in the classical Discrete Fourier Transform (DFT), the QFT is an important quantum operation. Fundamentally, the QFT is the quantum version of the discrete fourier transform. It is a linear transformation of qubits, moving from one quantum basis to another. In this case we are moving from the computational basis to the phase basis as will be discussed in more depth in section 3.2. There are many different applications for this basis transformation but we will be focusing on its ability to aid in the accumulation of the partial products generated during multiplication through phase rotations.

## 2.3 Noise and Depth in Quantum computers

As mentioned, there is "noise" present in quantum computers that leads to a lower accuracy when running a given algorithm. One of the most basic sources of this noise is decoherence, which is the interaction of the quantum system with the external environment. This usually results in the degradation the stability of a complex system and will cause the off-diagonal components that are present in an entangled state to disappear. Once this has occurred the system will have returned to a classical equivalent state. There is still much work to be done in reducing this source of noise including isolating the system and reducing the overall energy present that could interact in an undesired way with the system. Other significant sources include noise introduced by gates, inaccurate measurements, and cross-talk between qubits. All of these factors reduce the time that quantum operations can be reliably performed, leading to inaccurate results. The longer that the algorithm runs, the greater the impact of noise on the system due to the way that noise compounds on a system over time.

The algorithms run on a quantum computer are generally repeated a number of times, each time storing the resulting measurement before running again. These outputs can be viewed using a histogram to determine which occurred most commonly, which in most cases would be the desired answer. As you decrease the depth or how many gates need to be applied, there is less time for the noise to impact the system, thus resulting in a more accurate result. In terms of a histogram, there would be a focused probability distribution, leading to a more apparent solution from the algorithm. As our systems are built with more noise-dampening technologies, the impact of noise on the algorithms will reduce greatly. When the time comes that systems are noise free or adjacent, there will still be a need for improved depth, as it will lead to more efficient algorithm execution.

## 2.4 Related Works

Research into multiplication algorithms in quantum computing is a growing domain. With the publicity that this field has garnered in recent years, many are making progress in new applications and algorithms, multiplication included.

Another proposed method for multiplication utilizes similar quantum strategies to those proposed in this work [8]. The QFT is utilized to perform multiple weighted additions to calculate the product of two inputs. The focus of this work, however, was more on the implementation of a weighted average algorithm. As such, it does not go into depth on the gate-based implementation or depth predictions of its generated circuit. The algorithm developed in this work was directly implemented within Qiskit [14] as a callable function.

Research is also being performed on quantum implementations of classical very large number multiplication techniques. From the Karatsuba Algorithm [15][16] to the Schonhage-Strassen Algorithm [17]. While there are implementations that do perform these algorithms with correct outputs, the design of the quantum circuits utilizes a larger number of qubits and gates than is available on current and near-future quantum hardware. While these may not be usable on current hardware, they will be extremely important in future work for efficient large-number multiplication.

It is important to note that quantum multiplication algorithms are still in their early stages, and while those introduced may not be optimal in terms of efficiency, they are still important steps along the way to develop the most efficient implementation.

### 2.4.1 Computational

The Computational Basis (CB) method that will be discussed within this work is an extremely naive approach that does not take advantage of any quantum properties. This method leaves the operands and product in their computational basis and performs the same logical series of operations as it is performed on the repeated addition multiplier using analogous quantum gates to the classical gates. This design was first introduced by Vedral et al. [18] and referenced by Draper [19]. It utilized a combination of carry and sum gates built from controlled-not and Toffoli gates as shown in Fig. 2.3 and Fig. 2.4.

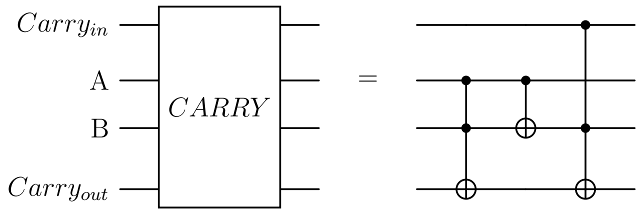


**Figure 2.3:** Computational basis implementation of the Carry operation on a quantum computer using controlled-NOT and Toffoli gates

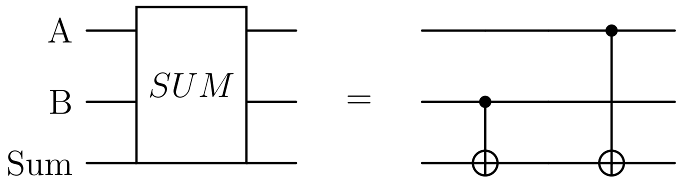


**Figure 2.4:** Computational basis implementation of the SUM operation on a quantum computer using controlled-NOT and Toffoli gates.

The controlled-NOT or CNOT gate is the controlled version of the Pauli-X Gate. The Pauli-X gate is analogous to the classical NOT gate and in this case is used to flip the state of the target from $0 \mapsto 1$ or $1 \mapsto 0$. The controlled version will only perform the NOT operation when the control qubit has a value of '1'. The toffoli gates used are also just extensions of the CNOT gate in that they are doubly controlled-NOT gates, which act only when each of the controls are '1'.

These operations are combined to perform the addition of two n-bit numbers, which is then further expanded in the form of a controlled multiplier [18]. Achieved by repeatedly performing the addition a number of times equal to the multiplier input for the computation. This design is not intended to be an efficient one, but rather to show the impact that taking advantage of quantum properties can have on the performance of the algorithm.

### 2.4.2 Repeated Addition

One of the basic and proven methods of multiplication in the quantum domain also uses Repeated Addition (RA) to compute a product [6][7]. This technique places the product register into the phase domain using the QFT and then uses phase rotations to repeatedly add the multiplicand. Each cycle the multiplier is decremented and measured. In the case where it is not equal to 0, the addition repeats on the product register. Once the multiplier has reached 0 the product is taken out of the phase domain by reversing the QFT and finally the result is measured. A basic circuit multiplying two 1-qubit numbers is shown in Fig.2.5 to better visualize the order that operations are performed.

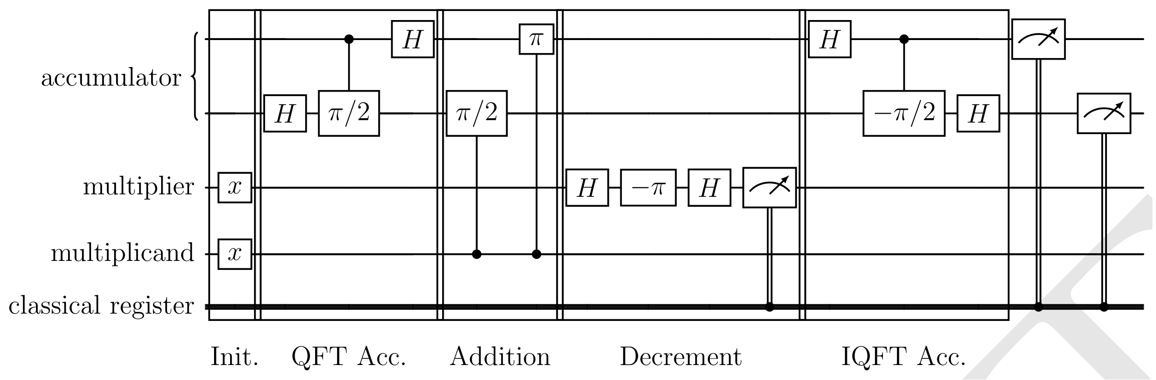


**Figure 2.5:** Full 1x1 multiplication circuit utilizing the repeated addition approach to multiplying two numbers. In this example the multiplier was initialized to '1' but if it was larger then the addition and decrement stages would repeat until the multiplier was reduced to 0.

While rudimentary, this method has been shown to work in both simulation and smaller implementations on physical quantum computers. There is an additional benefit in that for small input sizes this will need to repeat very few times resulting in small execution times. Due to these factors and its reproducibility, this method will be used as a point of comparison between different multiplication techniques.

### 2.4.3 Quantum Fourier Multiplier

One additional method for multiplication in the quantum domain, the Quantum Fourier Multiplier (QFM), utilizes a weighted addition to scale the basic addition operation used in the repeated addition by a weight that is determined by bit significance [8]. This algorithm also uses the QFT on the product register, and performs additions of the multiplicand onto the product. The main difference is that each bit of the multiplier is used as a control for one of the additions performed. The significance of the multiplier bit determines what weight is applied to the addition on the product register scaling from $2^0$ to $2^{n-1}$ for an n-bit multiplication. The least significant addition performed in the case of a 2-bit by 2-bit multiplication is shown in Fig. 2.6

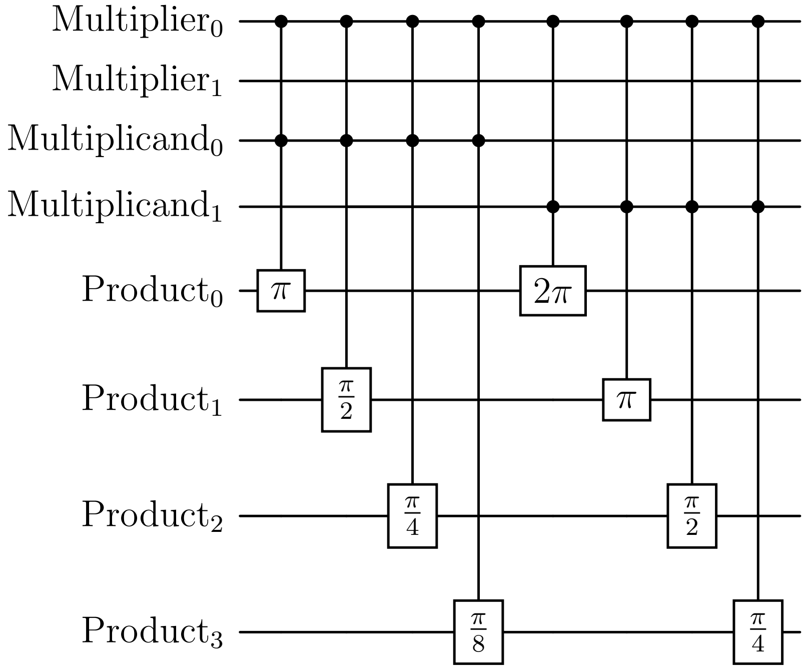


**Figure 2.6:** An example of the least significant quantum fourier addition operation showing each phase shift for a 2 bit by 2 bit multiplication. In this case the least significant bit of the Multiplier is controlling the significance of the phase shifts being applied.

The phase applied to each of the product bits in each addition performed is determined by a combination of the significance of the multiplier, multiplicand, and product using eq. 2.3, where s and j are the current index of the multiplier and multiplicand respectively and n is the current index of the product.

$$R = \frac{\pi i}{2^{n-(j+s)}} \tag{2.3}$$

This method is similar to that studied in this work but includes inefficiencies in the application of the phase shifts that will be discussed in more depth in the design of the QAM. This algorithm is one which IBM Qiskit has created a function for to multiply two binary integers and as such will be an important comparison as one of the current leading resource efficient multiplication techniques.

# Chapter 3

**Quantum Array Multiplier**

## 3.1 Classic Array Inspiration

This work proposes a new approach to performing multiplication in the quantum domain based on the principles of the QFT and inspired by the classical array multiplier. In the classical design, the array multiplier can be separated into distinct stages. Once these stages were outlined, they were then converted to the quantum domain and improved through the use of quantum properties that allow the parallel execution of stages without relying on the carries from previous stages.

In the case of the first row there is no input coming from a previous stage so it is implemented using only AND gates between the least significant bit of the multiplier and each bit of the multiplicand as shown in Fig.3.1.

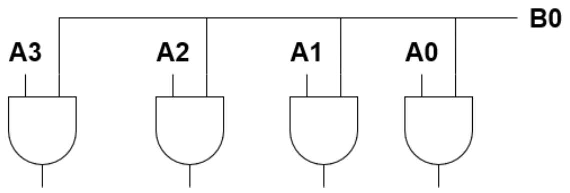


**Figure 3.1:** The first row if the classical array multiplier. Utilizing 4 AND gates the least significant bit of the multiplier operand is paired with each bit of the multiplicand, outputting a 1 in each column only if both inputs are high.

As more significant bits of the multiplier, in this case 'B', are used in following stages the significance of the result increases as well. In the example of the first row the addition between 'A0' and 'B0' will only impact the least significant bit of the product. When we move to the second row, as shown in Fig.3.2, the addition between ‘A0’ and ‘B1’ will impact the next bit of significance. Continuing this pattern the significance of each operation is determined by the significance of the inputs, ‘Ax’ and ‘By’ will directly add to the product bit with an index of (x+y).

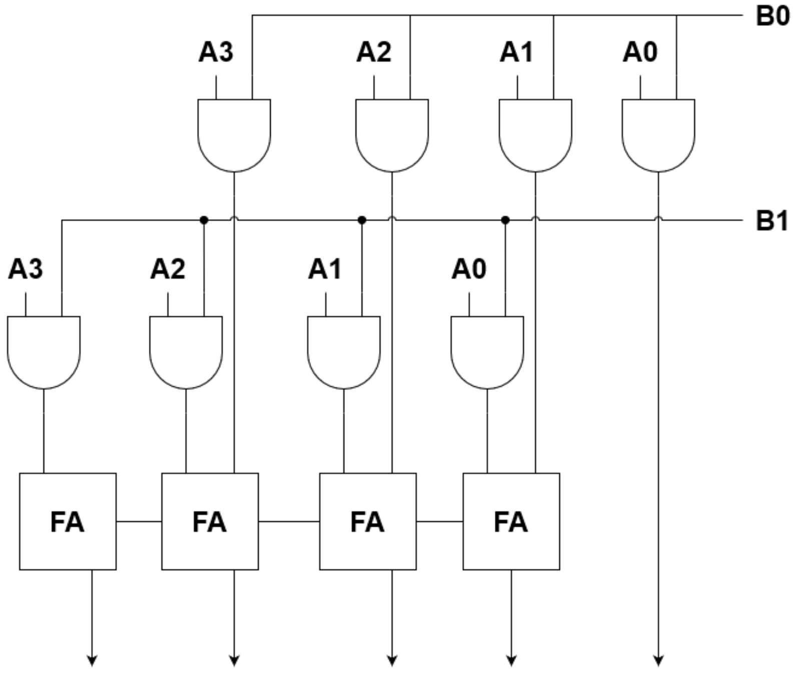


**Figure 3.2:** The first and second row, now with the full adder component included. The same set of AND gates are used for the second row for the partial product, then the result from the previous row is passed into the full adder which handles any carries between the columns. This pattern continues for each row going forward.

An additional component included in the second row and beyond is the FA. This allows a carry from the previous column to propagate through each row of FA’s, while the sum moves to the following row or product bit if there is not a FA under it. This allows any addition overflow to roll over to the next most significant bit and is a critical component to a functional array multiplier. The organization of the second row repeats until there are as many rows as there are bits in the multiplier. The important components present in this design, needed for its implementation on a quantum computer, are the AND gate and the FA.

## 3.2 QFT and IQFT

The use of the QFT was prompted by the poor scalability of the a strictly computational basis implementation. Table 3.1 [1] summarizes the different ways in which information is represented in the computational basis vs. the phase domain through QFT, for a two qubit example. The information in the QFT case is encoded in the relative phase of each basis state. For example, while the $|00\rangle$ state gets transformed into a state where there is no relative phase difference between the different basis states, $|01\rangle$ has a $\frac{\pi}{2}$ relative phase on $|01\rangle$, $\pi$ relative phase on $|10\rangle$, and $\frac{3\pi}{2}$ relative phase on $|11\rangle$ ($\frac{\pi}{2}$) increments), which is also represented in a $\pi$ phase difference on the first qubit, and $\frac{\pi}{2}$ on the second. As one moves down the table, the last column evolves by rotating each qubit an amount determined by eq. 3.1 where n is the number's decimal value, N is the number of qubits used for the representation, and $s$ is the significance of each individual qubit. In the case of a three qubit representation, the third qubit would change its relative phase at $\frac{\pi}{4}$ intervals.

**Table 3.1:** Quantum Fourier Transform break down for two qubit case. When the state is placed in the phase domain, this means that the information is encoded in the relative phases of the states: $n$ column represents the classical decimal value encoded in the quantum state, *CB* column represents the computational basis encoding of this value. *QFT (states)* represents the Quantum Fourier Transform of this computational basis encoding, and next to it, *QFT (qubits $q_1, q_0$)* represents the state of each qubit in that global QFT state, in this case with two qubits of significance $q_s$.

| n | CB | QFT (state) | QFT (qubits $q_1, q_0$) |
|---|---|---|---|
| 0 | $\|00\rangle$ | $\|00\rangle + \|01\rangle + \|10\rangle + \|11\rangle$ | $(\|0\rangle + \|1\rangle)(\|0\rangle + \|1\rangle)$ |
| 1 | $\|01\rangle$ | $\|00\rangle + i\,\|01\rangle - \|10\rangle - i\,\|11\rangle$ | $(\|0\rangle - \|1\rangle)(\|0\rangle + i\,\|1\rangle)$ |
| 2 | $\|10\rangle$ | $\|00\rangle - \|01\rangle + \|10\rangle - \|11\rangle$ | $(\|0\rangle + \|1\rangle)(\|0\rangle - \|1\rangle)$ |
| 3 | $\|11\rangle$ | $\|00\rangle - i\,\|01\rangle - \|10\rangle + i\,\|11\rangle$ | $(\|0\rangle - \|1\rangle)(\|0\rangle - i\,\|1\rangle)$ |

$$\varphi = \frac{n\pi}{2(N-s)} \tag{3.1}$$

[1]QFT implementations requires swapping the qubits at the end. Misalignment between this table and the circuit implementation showing reverse order is due to this swapping stage, but implementations are correct in all cases.

Looking at Table 3.1, it can be noticed that adding two numbers can be achieved rotating each individual qubit by the corresponding relative phase. For example to take the state from $n = 1$ to $n = 3$, one would simply apply a rotation of $2\pi = 0$ to $q_1$'s relative phase, and $\frac{2\pi}{2} = \pi$ to $q_0$'s relative phase. This would be equivalent to an addend $a = (2)_d = (10)_b$, each of the addend's bits is represented by $q_a$, where $a$ is the significance of each of the addend's qubits. It can be said more generally that controlled rotations of a phase determined by eq. 3.2 occur on each result qubit, controlled by each of the addend's qubits.

$$\varphi = \frac{2^a \pi}{2(N-s)} \tag{3.2}$$

In order to place a given accumulator's qubits into the phase domain to perform addition, a QFT operation is performed to change the basis of the quantum state, according to Equation 3.3.

$$QFT = \frac{1}{\sqrt{N}} \sum_{j,k=0}^{N-1} e^{\frac{i2\pi jk}{N}} |k\rangle\langle j| \tag{3.3}$$

The QFT operator applied to the accumulator transforms its representation from the computational basis $|k\rangle$ to the phase basis $|j\rangle$. An example of a two-qubit QFT quantum gate implementation is shown in Figure 3.3. Table 3.1 shows the results on the quantum state of this transformation —column *QFT (state)*—. The circuit accounts for the phase rotations necessary for both qubits to transition into the phase domain through the use of a controlled rotation after the first Hadamard gate. While this operation is relatively simple, its depth quickly increases as more qubits become necessary for multiplication.

Similar to the QFT operator, an IQFT operator is used to change the basis of each qubit of the accumulator from the phase domain back to the computational basis, allowing for the result of the multiplier to be measured into the classical bits.

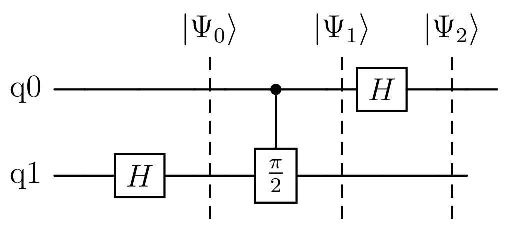


**Figure 3.3:** Quantum Fourier Transform Stage.

The IQFT is performed as described by Equation 3.4. Similar to the QFT circuit it utilizes two Hadamard gates as well as a controlled phase shift, though in this case, the phase shift is negative rather than positive. IQFT has the same depth as the QFT, which contributes to the rapid growth in depth as bit width increases.

$$QFT = \frac{1}{\sqrt{N}} \sum_{j,k=0}^{N-1} e^{\frac{-i2\pi jk}{N}} |j\rangle\langle k| \tag{3.4}$$

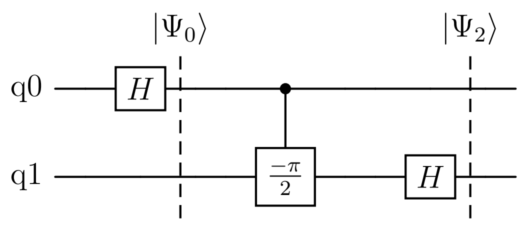


**Figure 3.4:** Inverse quantum fourier transform stage.

## 3.3 Quantum Array Multiplication

The efficiency improvement over existing quantum multipliers comes from two sources: the efficient use of addition on the phase domain —which utilizes a quantum property—, and the classical hardware design of array multipliers. Speedup is shown when the unique capabilities of quantum computers are utilized, though, not every algorithm can take advantage of these capabilities to see speed up on a quantum system.

### 3.3.1 Initialization

The first stage for the multiplier is to set the qubits into their desired positions. There are 3 main components that need to be initialized for this algorithm to work properly. The first two are the multiplier and multiplicand registers. There are a number of qubits in each register determined by the bit width of the correlated input. For every binary 1 encoded in either input an "X" gate is placed on the corresponding qubit to flip it from the initial 0 state into the 1 state as shown in Fig.3.5. The other component is the product register. For this work the product register is automatically initialized to a value of '0' and will be used as such, but if a sum of multiplications is desired then this register does not need to be reset to 0.

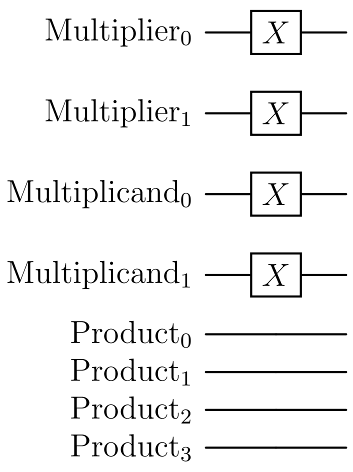


**Figure 3.5:** Initialization stage for the Quantum Array Multiplier. In this example both the multiplicand and multiplier are being initialized to decimal values of 3 using Pauli-X or NOT gates. These take the initial state of '0' and flip it resulting in a '1' stored in each of the qubits.

This stage only takes one time step to accomplish as each of the "X" gates can be run at the same time. Once they have been run the two multiplication inputs will have successfully been encoded into the quantum system and the multiplication can begin.

While it was not the focus of this work it can be noted that if the number of result qubits is restricted to a value $n$, multiplication modulo $2^n$ is performed. This can be useful both when the modulus operator is an important part of an algorithm, but additionally as a method of reducing qubits used when higher order product bits are unnecessary for the function implementing this multiplication.

### 3.3.2 QFT

Using the QFT discussed in 3.2, the product register is transformed from the computational basis to the phase basis as shown in Fig.3.6. [2]

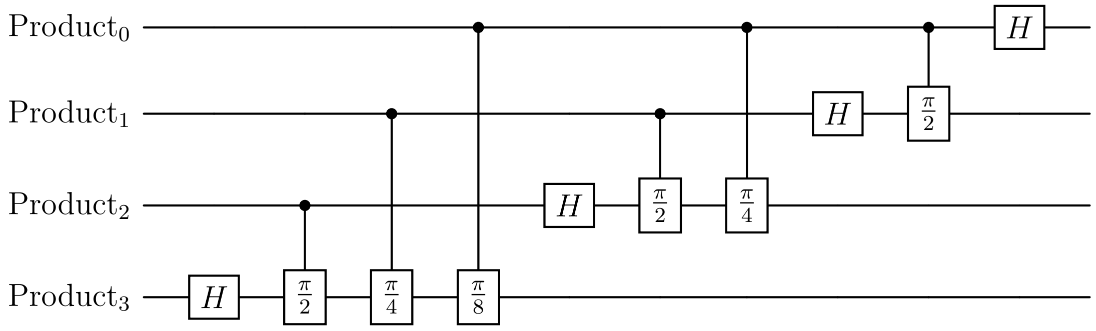


**Figure 3.6:** Quantum Fourier Transform stage changing the product register from the computational to the phase basis.

This moves the product qubit register from the computational basis into the phase basis. In order to perform the phase rotations integral to this algorithm the product register has to be in this state.

### 3.3.3 Array Multiplication

In order to implement the array multiplication in the quantum domain the two main components of the classical version, the AND gate and the FA, needed quantum counterparts. Due to the fact that addition is performed while in the phase basis

[2]If all qubits in the product register are 0 then the phase shifts in the QFT operation can be removed leaving only the Hadamard gate acting on each qubit. This can, especially with large inputs, reduce depth greatly but will not be included in the analysis to see more realistic applications of the algorithm.

using phase shift gates, the AND can be substituted using a doubly-controlled phase shift gate as shown in fig.3.7.

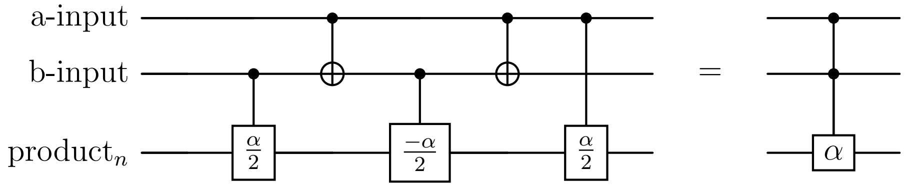


**Figure 3.7:** Double control rotation decomposition.

by using single controlled phase gates in this way we are able to ensure that a phase shift of $\alpha$ only occurs if both inputs are high. In the case that only 'a' is high, b will be inverted applying a negative phase shift. After which 'b' is un-inverted and 'a' applies a positive phase shift negating the previous phase shift. If only 'b' is high then it will apply both a positive and negative phase shift, causing no affect. If both are low then none of the controlled gates function and nothing happens. Finally if both are high then two phase shifts of $\alpha/2$ are applied equalling one full phase shift of $\alpha$ as intended.

This gate must be applied to the product qubit of the index determined by the two input qubits, as well as all qubits of greater significance than that index. The least significant phase gate of the current set of inputs shifts by $\pi$. Each of the following gates applied are divided by 2 following the form $\alpha = \frac{\pi}{2^n}$ as shown in Fig.3.8 where n increases by 1 for each qubit of greater significance acted upon. It can be noted that these phase shifts get exceedingly small when working with extremely large inputs, which is the problem targeted in section 3.4.

Now that the AND gate has been implemented from the classical design, only the FA is left. This is a very easy component to add as it is already handled with the additional phase shifts as part of the AND gate implementation. On a classical system the product is only reached after taking each of the previous stages and passing them through their respective FA's to see if there is any carry and sum. The difference in

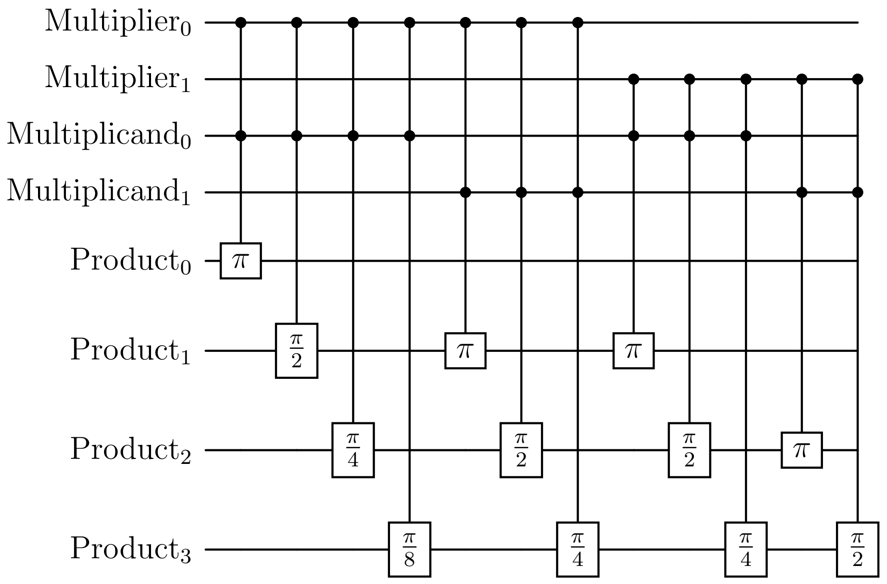


**Figure 3.8:** Quantum Array Multiplication stage showing each phase shift for a 2 bit by 2 bit multiplication.

the quantum implementation is that each addition is acting directly onto the product register independent of each other. When classical addition in a FA would result in a carry, the phase shifts will have rotated the next most significant qubit by $\pi/2$ twice effectively adding one to that qubit. Meanwhile the original qubit will have had two $\pi$ rotations, rotating the phase around the unit circle back to 0 where it started, effectively performing the carry that the classical FA performs.

Another benefit of the additions not depending on the outputs of the previous stages is that these operations, when interacting with different qubits, can be run in parallel with each other as determined during circuit optimization. This allows for the potential of slight speedup and, while not the intended benefit of this work, is nonetheless beneficial.

### 3.3.4 IQFT

After all the stages of the array multiplication are performed, the product is ready to be measured or used in another computation. Using the Inverse Quantum Fourier

Transform (IQFT) we can then take the product register back out of the phase and into the computational basis. This allows for us to either measure it or continue to use it as a continuation of another algorithm or process.

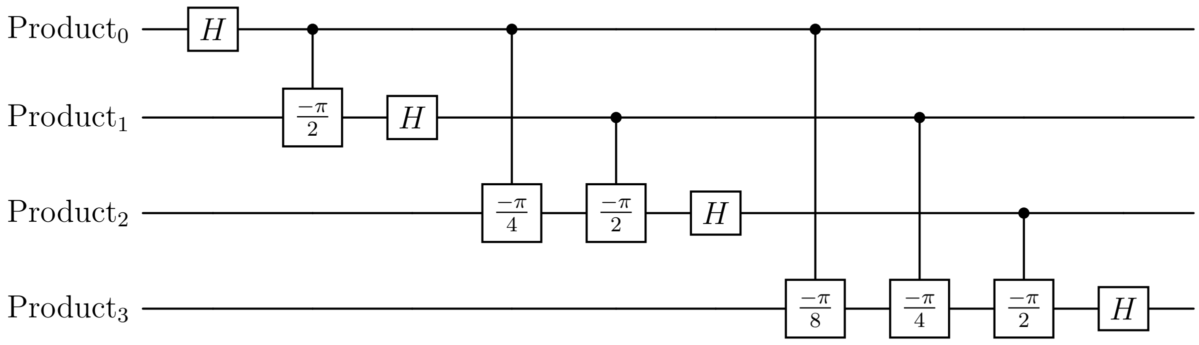


**Figure 3.9:** Inverse Quantum Fourier Transform stage changing the product register from the phase to the computational basis.

### 3.3.5 Measurement

Finally the measurement of the product qubits is performed to read out the result of the multiplication. The way the measurement is performed is dependant on the quantum computer utilizes, but in the abstract quantum circuit form it follows the same format. For each measured qubit in the system there is a classical bit that is assigned the value measured from the qubit, forming a classical register. By reading out the classical register one can tell what state the measured qubits resulted in, which in this case is the product of the multiplication that was performed.

The full circuit with each of the stages combined is shown in Fig.3.10. In this specific example it is multiplying 3 by 3 as each qubit of the inputs are being set to '1'.

## 3.4 Approximate Phase Multiplication

As the input size increases, the minimum phase shift also begins to get increasingly small. As they continue to get smaller and smaller, the rotation matrix that defines

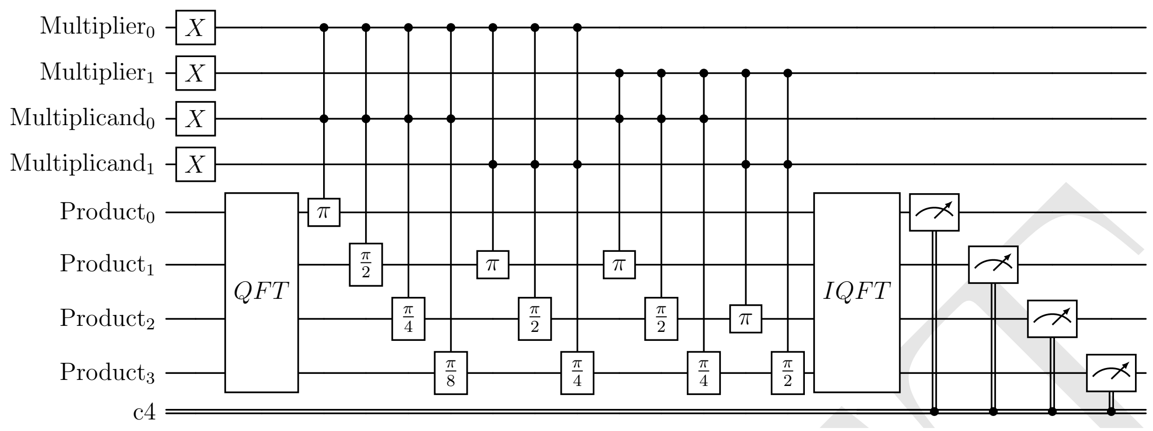
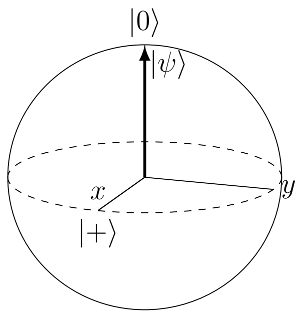


**Figure 3.10:** Quantum Array Multiplier (QAM) with each stage included starting with the QFT stage, followed by each of the row additions and finally the IQFT.

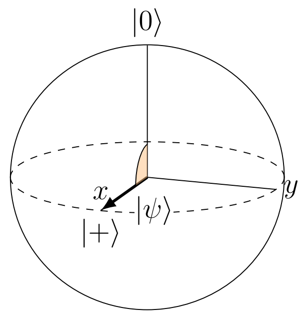
the phase shift begins to reach the identity. There are a few different issues that could arise from this. One such issue arises when reaching a point where we are no longer able to perform phase rotations below a certain threshold due to hardware limitations. Another issue with these small rotations is their impact on accuracy. Every gate inherently adds some amount of noise to the circuit. When the phase incurred by the gate has less of an effect on the computation than the noise accrued during that application, it may not be worth it to use the gate at all. Barenco, et. al. showed that in the case of the QFT an approximate version, the AQFT, could be more accurate than the full QFT[20]. The implications of this work are that the QFT and IQFT stages of the multiplier are able to be reduced by limiting the phase shifts to roughly $\lceil Log_2(Product\_width) + 2 \rceil$. The reduction in the number of gates applied is more easily noticeable on circuits with larger input sizes.

Approximation in the phase domain does not affect to output like it would when precision is dropped in the classical setting. To better illustrate the effects of approximation, a one-qubit example is shown in Figure 3.11. Through this example, we intend to show how small rotations in the phase domain (whether added or removed) have a small impact on the measured outcome of the qubits. In this case, the rotation is added rather than removed. As shown in Figure 3.11, the qubit is initially in state

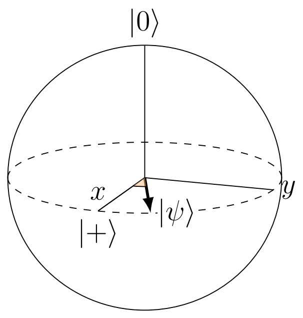
$|0\rangle$. After applying a Hadamard gate, it will evolve to the $|+\rangle$ state, now placed in the phase domain. To simulate the impact of the approximate phase shifting technique a (small) phase of $\pi/8$ is applied, taking it out of the $|+\rangle$ state. After the small rotation, a Hadamard gate is once again applied to return the qubit to the computational basis. If there were no approximation, then the Hadamard would take it directly back to the $|0\rangle$ state, but with the small additional rotation, the final state is not exactly set to $|0\rangle$. Still, a measurement on the computational basis will project this state to the $|0\rangle$ state "most of the time" ($p_c = 0.96$) this case with probability and only sometimes it will wrongly measure the $|1\rangle$ ($p_w = 0.04$). This is the reason why even when rotations are removed, it is still possible to measure the correct output with high probability.



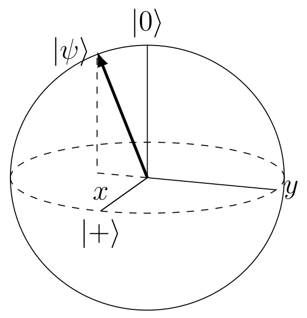
**(a)** Initial State



**(b)** Hadamard Applied



**(c)** $\pi/8$ rotation



**(d)** Second Hadamard

**Figure 3.11:** Example Bloch diagram transformations showing the impact of approximation in the quantum phase basis. Starting in (a) with the initial state of $|0\rangle$ a hadamard is applied transitioning to $|+\rangle$ as shown in (b). Once in the phase basis a sample rotation of $\pi/8$ is applied, (c), followed by a Hadamard gate application returning the state almost to $|0\rangle$ as shown in (d).

The idea of the AQFT is taken a step further by Draper[19] which proposes using the same technique as the approximation of the QFT, but this time applied to the phase rotations of the quantum addition being performed. By eliminating the phase additions that fall below a certain threshold, a reduced number of gates can be added, and using the same logic as the AQFT, in the presence of noise or decoherence, there is a potential increase in the accuracy of the circuit. Additionally highlighted by Draper was a method for parallelizing the execution of the addition operations in which each addition may be performed in $log_2 n$ time slices.

Both of these techniques were utilized within this work in the design of the AQAM. This design follows the same structure as the normal QAM, but includes a phase limit determined using the qubit size of the product register.

# Chapter 4

**Results**

## 4.1 Summary of designs

Following the explanation above (Section 3.4), four designs of a quantum multiplier were implemented and compared:

- Computational Basis (CB): For comparison, a naive approach of multiplication through repeated addition is also performed on the computational basis as opposed to the phase domain, through the use of Toffoli and CNOT gates.

- Repeated Addition (RA): This design implements multiplication of two integers by repeatedly adding the value of the multiplicand into the accumulator as many times as indicated by the multiplier. The multiplier is decremented towards a zero value that indicates the end of the computation. This implementation also requires the measurement and comparison of the multiplier during runtime.

- Quantum Fourier Multiplier (QFM): The Quantum Fourier multiplier also takes advantage of the phase, but instead of performing repeated addition it utilizes weighted addition.

- Quantum Array Multiplier (QAM): For the new approach inspired by the classical array multiplier, double controlled rotations realize the bit multiplication and consequent addition.

- Approximate Quantum Array Multiplier (AQAM): Taking the same approach as the QAM, this method additionally reduces the number of phase shifts by implementing a threshold defined using the product register size.

### 4.1.1 Qiskit

In order to construct these quantum circuits, IBM's open-source software development kit, Qiskit[14], was used. Qiskit provides the ability to construct and test quantum circuits and algorithms on simulators as well as actual hardware. With each of the algorithms compared and proposed, quantum circuits were implemented and simulated with and without noise on the Qiskit Aer Simulator [14]. The results were always correct in the non-approximate implementations when simulated without noise. Simulations were run for both square and identity multiplication using classical input sizes ranging from 1 to 12 bits when possible. With the larger test cases, hardware limitations led to impractical simulation times, and as such, some cases beyond a certain threshold are excluded. As per performance, since the execution was not performed on real hardware, we present the depth and simulated accuracy of the circuits as a metric of efficiency and an approximate metric of compared performance for all different designs. This depth was provided by Qiskit's depth function. Therefore, it does not include swap gates to match any specific hardware layout; it is simply a reflection of the number of computational quantum layers necessary for completion.

## 4.2 Circuit Analysis

### 4.2.1 Width of quantum multipliers

All five compared quantum multipliers use the same number of qubits for the same size operands. Therefore, the width is $2(m + n)$ where m and n are the size in bits (qubits) of the two integer operands. As they are the same in each of the algorithms being compared, the number of qubits is not necessarily a limiting factor for the implementation of the integer multiplier. Additionally, it is not a factor that can be improved across the proposed designs since it remains equal or very similar for all of them.

### 4.2.2 Depth of quantum multipliers

Depth is a critical parameter in the design of any quantum circuit. At the time of this research, the correctness and efficiency of these circuits were only verified through simulation, and as such, this research does not provide performance metrics on real hardware. Depth, however, defined as the number of computational steps necessary to completion, is a first approximation to the performance and performance improvements that one design displays over the other.

Depth additionally depends on the parallelism that the computation is able to extract out of the gates acting on their corresponding qubits. If two gates are acting on independent qubits, then both of those can be performed at the same time, and as such, the value added to depth is 1 rather than 2. The quantum array multiplier approach uses parallelism and quantum phase computation to significantly improve the depth of the design. On one hand, the computation on the phase domain and the double-controlled rotations to implement the AND-adder combos make efficient use of the quantum properties. Rotations can accumulate directly on the product qubits, avoiding information exchange in between "rows," and there is no need to pass carry information. In addition, parallelism is exploited since each controlled gate can act independently as long as it acts on separate qubits. For example, looking at Figure 3.10, the first double-controlled rotation (acting on qubits $Multiplier_0$, $Multiplicand_0$, and $Product_0$) can be performed at the same time as double-controlled rotations eleven or twelve (on qubits $Multiplier_1$, $Multiplicand_1$, and $Product_2$ or $product_3$).

On the other hand, as stated in Section 2.4.2, the repeated addition multiplier, only performed the additions demanded by the multiplicand, while the new proposed algorithm always performs the same number of additions for a given operand size. For that reason, the circuits' depth comparison depends on the numbers being multiplied. The comparisons are presented for the two extreme cases: the trivial Mx1 product and the square MxM product.

## 4.3 Depth Analysis

### 4.3.1 Square Multiplication

The first set of tests that were completed were the square multiplication tests. It took numbers, each with bit widths from 1 to 20 having every bit initialized to 1, and used them as both the multiplier and the multiplicand. This was done by initializing each the input qubit registers using X-gates. Having all bits initialized high can be regarded as the worst case scenario for execution as it leads to both the highest number of repeated additions as well as the most gate activations needed to perform the multiplication. Each gate that is executed attributes some amount of error to the system, so more gates incurred means more error introduced and vice-versa.

The resulting depths of running this test set on each of the different multiplication algorithms can be seen in Fig. 4.1.

Looking at the different slopes it is easy to identify the exponential scaling of the CB and RA algorithms. These two, most notably the CB, do not readily scale up to higher bit width multiplication as their depth quickly becomes too much for both simulation and accurate results. The exponential increase in depth comes from the number of repeated additions doubling each time the bit length of the inputs increases by 1

Of the other three cases there is a notably similar polynomial trend. The main

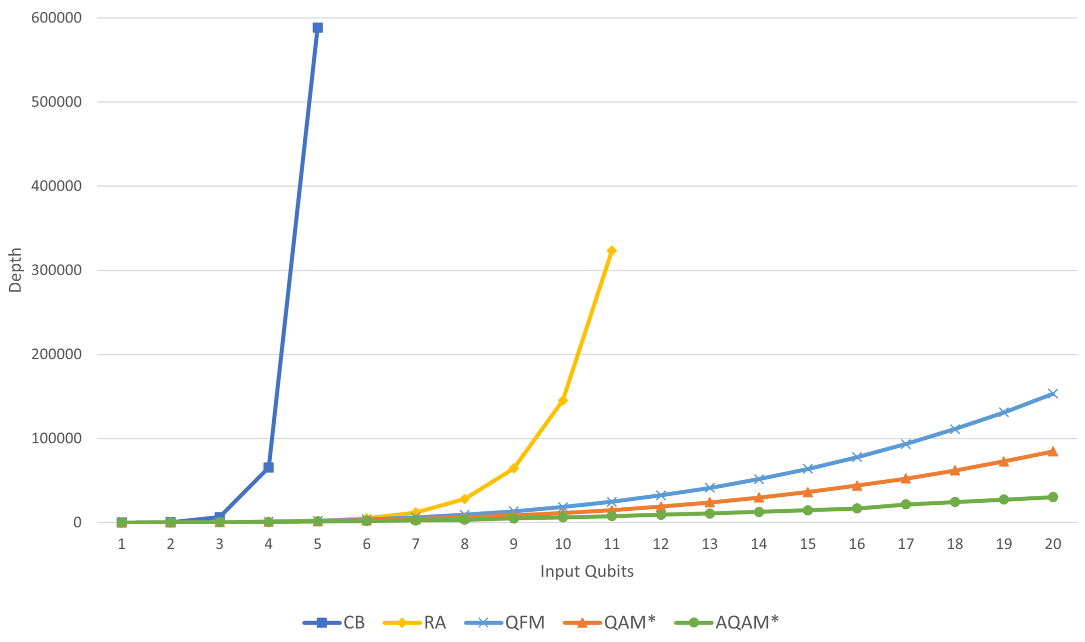


**Figure 4.1:** Depth results for the square multiplication of each of the 5 algorithms being compared. In order from worst to best scaling there is the CB, RA, QFM, QAM, and finally AQAM with the first two having exponential scaling and the last three being polynomial. Additionally a projected polynomial is presented for each of the QFM, QAM, and AQAM.

difference between the QFM and QAM depths comes from the efficient use of controlled phase shifts. The QFM algorithm utilizes phase shifts of ($2\pi$, $4\pi$, $8\pi$, etc.) when performing the weighted addition, which both does not fundamentally change the resulting values on the qubit being operated on, and adds unnecessary gate noise. Following the design of the classical array multiplier, the QAM ignores these operations by default as they are not part of the classical implementation. This would be analogous to flipping bits of lesser significance twice, returning them to the same value they had started in. The removal of these gates results in greatly reduced depth, especially when considering higher order multiplication where the number of these unnecessary rotations is almost equal to the number of necessary ones. Finally, the AQAM further reduces its depth by limiting the extremely small phase shifts which have relatively little impact on the final result of the multiplication. This reduction in depth does come at the cost of a loss of accuracy when compared to the non-approximate solution as shown in Fig. 4.4.

It can be seen that the AQAM takes on a stepping form in which there are certain points that the depth jumps slightly higher, not following the standard curve before returning to normal rate of increase. The rise in depth between the 16 and 17 qubit case comes from the calculation of the minimum phase allowed. When calculating the phase the minimum is $\pi/2^N$ where N is calculated with $\lceil \log_2(product\_width) + 2 \rceil$. In the 16 input qubit case the product register has 32 qubits resulting in $\lceil \log_2(32)+2 \rceil = 7$, whereas in the 17 input qubit case it is calculated to be $\lceil \log_2(34) + 2 \rceil = 8$. This increase in the value of N allows smaller phase shifts to be performed leading to the slight increase in accuracy at the cost of an increase in depth. Following this same logic there would a slight increase in depth at any stage between a value of $2^N$ and $2^N + 1$ for the same reason. This relationship for minimum phase was first given in [20] and built upon in [19]. A different relationship could be developed when systems are less noisy to change how the minimum phase shift impacts the accuracy by setting a smaller minimum phase shift than that which is currently utilized.

### 4.3.2 Identity Multiplication

The second case for comparison between the different implementations is identity multiplication. In each of these cases increasingly large values are being multiplied against 1 for a trivial example of multiplication. This test case provides an optimal state in that the minimum number of controls are being triggered leading to lower gate noise, and in the case of CB and RA there is a significantly lower depth incurred as they only need to repeat once. The resulting depths of this test set can be see in Fig.4.2.

Similar to the case of square multiplication, the CB implementation very quickly exceeds the other methods in terms of depth. This shows that even when performing a trivial test case it is not a viable method for multiplication on a quantum system,

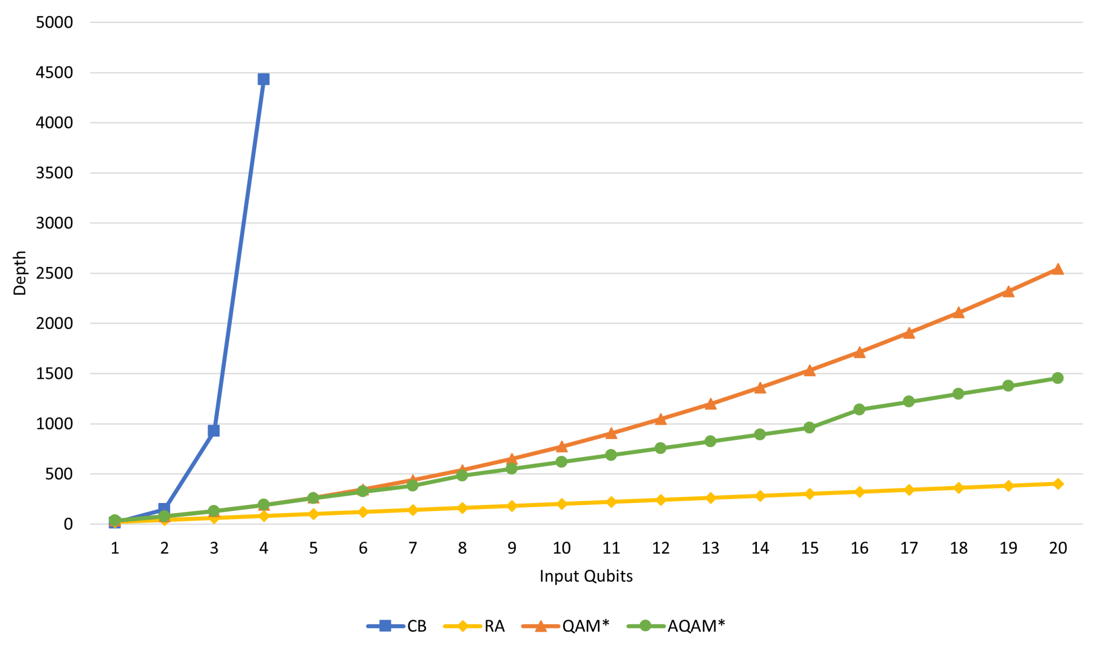


**Figure 4.2:** Depth results for the identity multiplication of four of the different algorithms. In order from worst to best scaling there is the CB, QAM, AQAM, and RA. In this case the RA is the shortest as it only needs to iterate once before finishing its multiplication.

which is to be expected as it was notedly a naive approach when it was first introduced. The RA method in this case outperforms the other algorithms as it only needs to perform each addition once to calculate the product of the trivial multiplication. This method shines when considering cases where one of the two multiplication operands is extremely small in relation to the other operand. Looking at the QAM and AQAM methods, they do fall behind RA due to the fact that they still need to form the entire array structure used in the multiplication. The benefit that still holds true for QAM and AQAM over RA is that they do not require mid-circuit measurements to operate. This may limit the operation of the RA algorithm on systems where mid-circuit measurements are either not possible or costly. In this test set the QFM was not able to be included as, per limitations of its implementation in Qiskit, it cannot be used in cases where the bit widths of the two operands are different.

### 4.3.3 Varied Input Multiplication

In order to better visualize how the depth changes between the trivial and square multiplication an additional set of tests was run. The third set of tests simulated how the depth of each circuit grows depending on varied input sizes. In all of the simulations the multiplicand used as an input was 12 bits long, each of which were initialized to '1' corresponding to a decimal value of 4095. The multiplier was then varied from 1 to 4095 by left shifting the input and inserting a 1 into the least significant bit. The results of which is shown in Fig. 4.3. Due to their structure not being influenced by the decimal values of the operands the QFM, QAM, and AQAM have a constant depth.

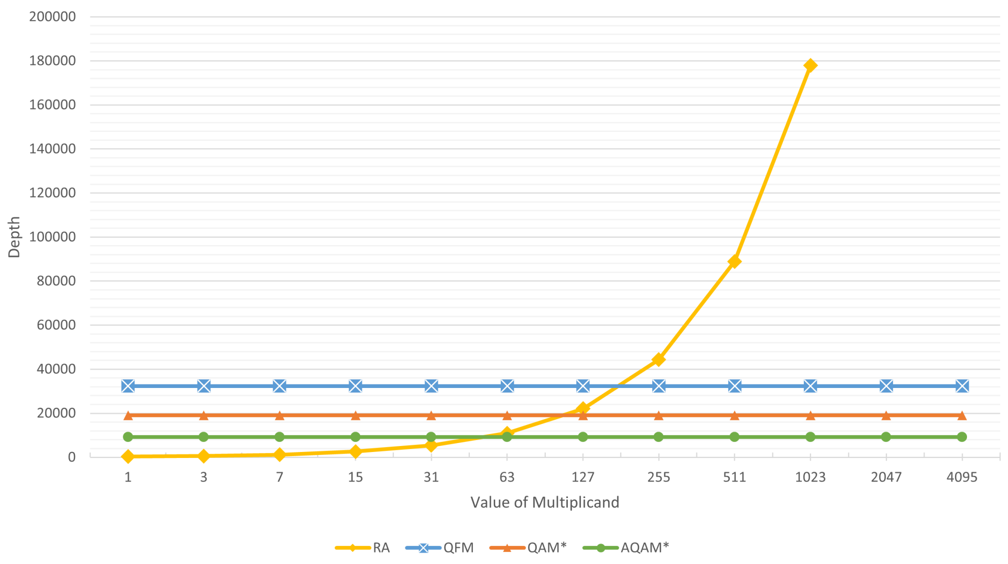


**Figure 4.3:** Depth of each algorithm given a multiplicand value of 4095 and a multiplier value scaling from 1 to 4095. The depths for the QFM, QAM, and AQAM all remain constant regardless of the input value due to their standardized structure. RA roughly doubles in depth at each interval as the number of times it needs to repeat has been doubled.

In the cases where the value of the multiplier is guaranteed or known to be small relative to the multiplicand the RA algorithm has better depth, and consequently accuracy. At an input value of 63, 127, and 255 it begins to perform worse than the AQAM, QAM, and QFM respectively. This shows that the repeated addition is only the desired choice for 1.5% of the possible input combinations when compared to AQAM, or 3.1% and 6.2% for QAM and QFM. When the input value is not known or guaranteed to be fairly large the ideal algorithms would be either the QAM or AQAM depending on the noise present in the system.

## 4.4 Simulated Accuracy

When simulating these circuits Qiskit[14] was used for both the ideal and noisy simulation models. In the case of the ideal simulation, there is no noise included in the calculation of the possible results making it similar to running on an ideal quantum computer. The noisy simulations include simulated noise from each of the primary sources as part of the qiskit aer simulation package. There are different models which are created to mimic real systems and provide an outlet to test your code without actual access to these systems. One model used to simulate noise in this work came from the GenericBackendV2. This was selected because it allowed for a variable number of input qubits rather than being limited to a small sample of qubits like those on real devices. The downside of this is that it has a high default noise model, so noise accrues rather quickly in each of the following simulation. An additional noise model was created with less noise than that of the default model for the GenericBackendV2 which was also used to show performance on systems with less noise present.

The accuracy presented within these experiments comes from analyzing the resulting histogram of the simulations. Each time a circuit is simulated it is executed and measured in 1024 shots. For example if one was measuring a qubit in perfect superposition between 1 and 0 it can be expected that roughly half the shots would return 1 and the other half would return 0. To calculate accuracy the number of shots that resulted in the correct output is calculated and then divided by the total number of shots. These results are limited to 7 input qubits for most of the implementations

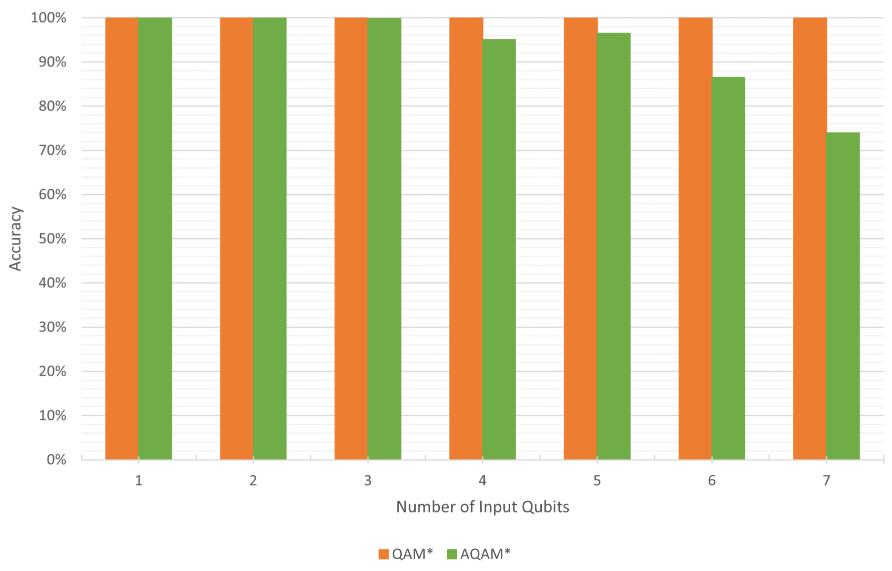


**Figure 4.4:** The ideal simulation of the QAM and AQAM implementations. As with all of the other algorithms, when in a noiseless environment, QAM has 100% accuracy for any given input width. AQAM sees a reduction in accuracy as the input size increases as it is removing an increasingly large number of phase shifts from being executed. This accuracy can be improved at the cost of depth by altering the function that determines the minimum phase shift to allow a larger set of phase rotations.

due to the hardware intensive nature of the executing each simulation. It was not practical to try to simulate beyond 7 qubits due to both execution times and the hardware required.

### 4.4.1 Ideal Simulation

The first set of tests were done using the ideal simulation in order to show how the AQAM reduces in accuracy as the number of qubits increases. The results of these simulations for square multiplication can be seen in Fig. 4.4.

Fig. 4.4 shows that, under ideal conditions, the QAM remains 100% accurate regardless of bit width while the accuracy of the AQAM begins to decline as more qubits are added. When it reaches an input qubit width of 7 the accuracy ends up being 74%. This accuracy loss is less pronounced in cases where not every bit is high in the inputs so this can be seen as the worst case accuracy for the given AQAM

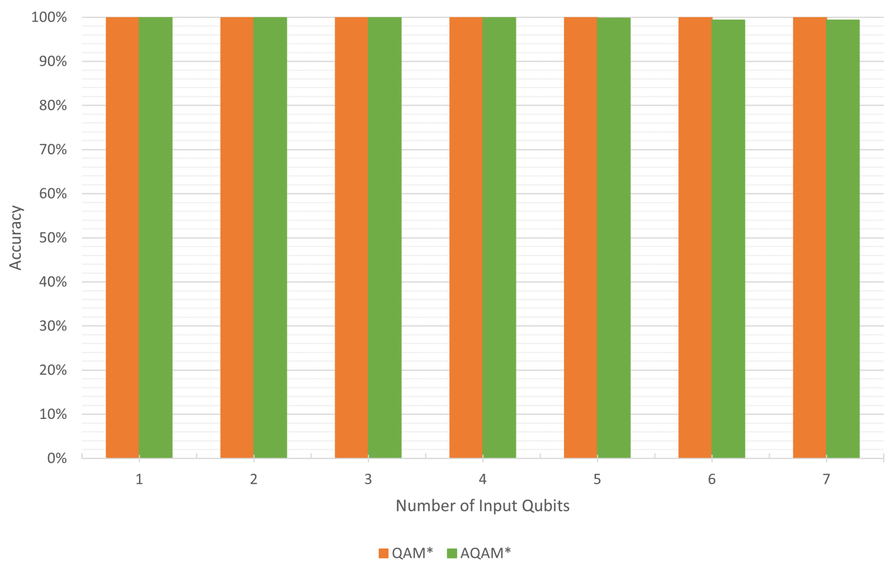


**Figure 4.5:** The ideal simulation comparing the QAM and AQAM implementations. As in the square multiplication case the QAM remains 100% accurate while the AQAM begins to fall, but in this case the accuracy of the AQAM falls much slower both due to its decreased depth and a lower count of gates incurred.

implementation. Additionally if the reduction in ideal accuracy becomes too great, it can simply be adjusted to allow smaller phase shifts to occur than the current limit. This will immediately bring the AQAM closer to the performance of the QAM. A comparison of the accuracy of the two implementations in the trivial multiplication case is shown in Fig. 4.5.

While there is still a reduced accuracy in the AQAM it has a noticeably lower loss than the square multiplication. This can be attributed to both to the fact that the identity cases have lower depth and to the lower number of phase shift activations that are performed in the trivial multiplication.

### 4.4.2 Noisy Simulation

Each of the different multiplication algorithms were then run in noisy simulations to see how their accuracy changes with increasing input width. The first set were run on what is a more realistic environment as shown in Fig. 4.6, with the exception of

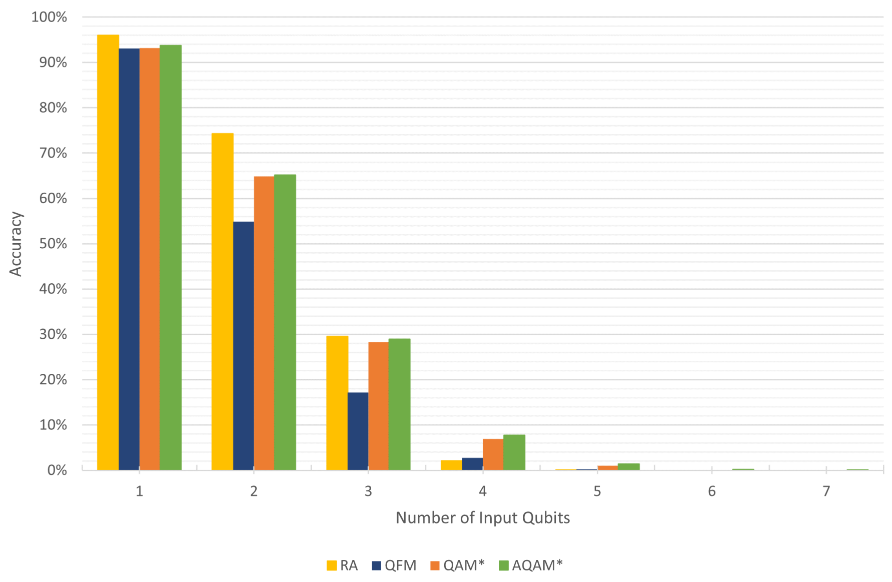


**Figure 4.6:** A noisy simulation showing the accuracy of each algorithm given an increasingly large input size. While initially the most accurate for an input width of one, the RA quickly falls below the others. Following this in terms of accuracy at higher bit widths is the QFM then the QAM and the AQAM. The difference between the QAM and AQAM shows that the AQAM has a slightly increased accuracy even with the reduced accuracy present in an ideal simulation.

the computational basis implementation due to its unreasonable scaling of depth.

In a similar vein with the depth graph, it can be seen that, while the RA implementation starts with the most accuracy, it is the first to reach near 0% accuracy at 4 qubits. The QFM does eventually perform better than the RA, but it remains worse than both the QAM and AQAM implementations. Finally, comparing the performance of the QAM and the AQAM, they are almost identical in accuracy, with the AQAM having slightly better accuracy once it is greater than 3 input qubits.

The performance of the AQAM follows the same form as the QAM, even with the lower accuracy in the ideal simulation case, because of the significant reduction in depth that comes from removing the smaller phase shifts. This confirms that, while working on a noisy system, there can be a benefit to reducing the depth at the cost of accuracy. This relationship is more pronounced in the simulations with a reduced noise model, as shown in Fig. 4.7.

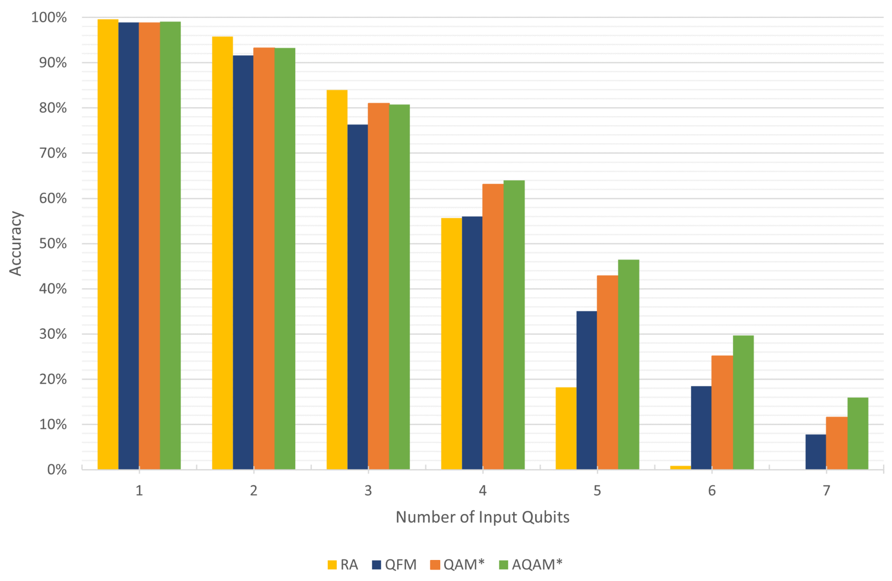


**Figure 4.7:** Results for a simulation which uses lower noise than the default backend. In this case the distinction between the algorithms is more apparent, with the RA still falling at a faster rate than the others even though it starts at the highest accuracy. Of the other 3 algorithms, in terms of worst to best accuracy, they are consistently in the order of QFM, QAM, and AQAM.

It is easier to see the potential benefit of the approximation technique in these results as the AQAM performs noticeably better than the original implementation of QAM. Both the QAM and AQAM outperform the other two algorithms once more than 4 qubits are utilized in the multiplication. Once the performance had been determined for square inputs, it was time to simulate the identity cases in noisy systems. The first case was performed once again using the more noisy backend, with the results shown in Fig. 4.8.

The simulated accuracy here shows that, in correlation to its lower depth, the RA operation has the best accuracy. Only needing to perform one addition for each of the sizes allows it to maintain a fairly high level of accuracy in comparison to the other techniques. The QAM and AQAM are both extremely similar as they have nearly the same depth, with AQAM beginning to have a noticeably lower depth once it reaches an input width of 5. The difference between the algorithms is further analyzed using a simulation in a less noisy environment, as seen in Fig. 4.9.

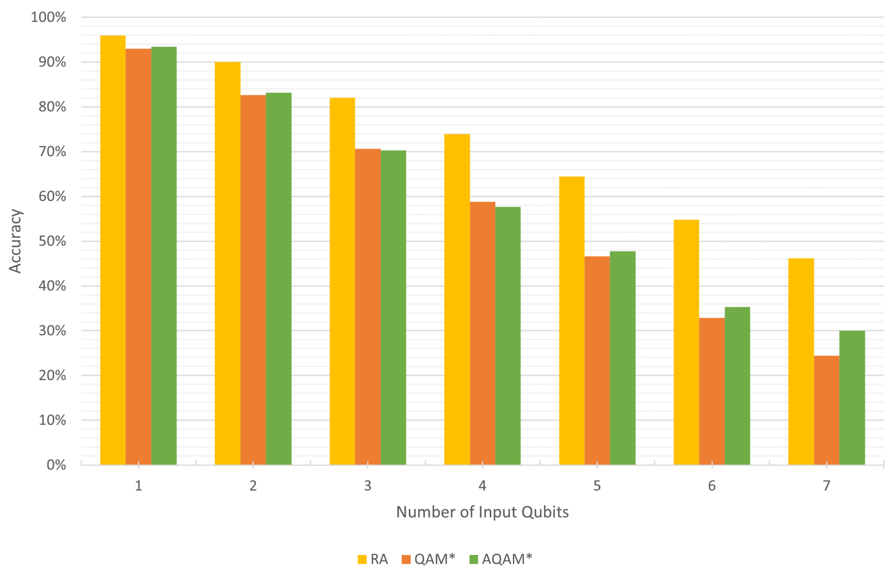


**Figure 4.8:** Simulation of trivial multiplication on a noisy system. Comparing the different accuracies in these cases shows that, in relation to its lower depth in identity cases, the RA outperforms both the QAM and AQAM in all cases.

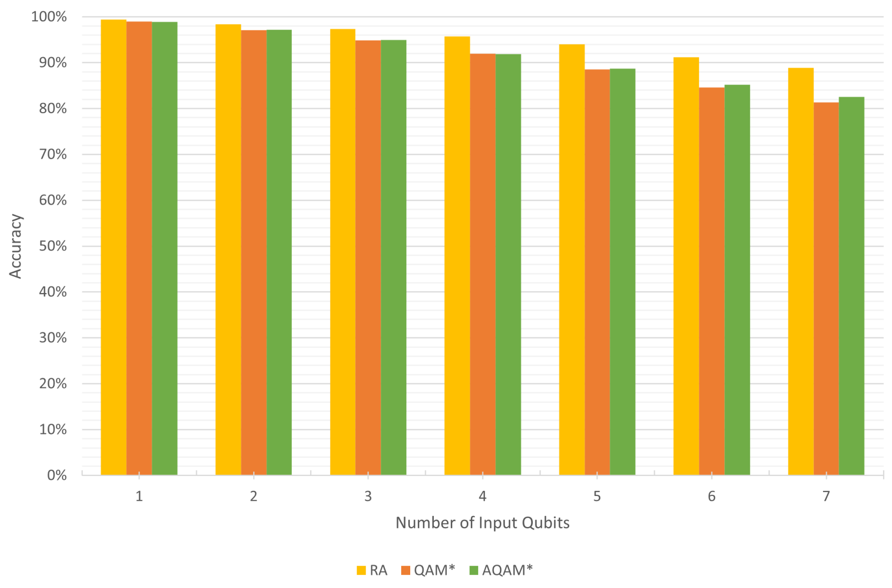


**Figure 4.9:** Similar to the noisy simulation, RA remains the best case algorithm for trivial multiplication taking advantage of having only one iteration required for the multiplication.

Both identity multiplication simulations follow the same trend as their respective depth relationships, which is the expected output in these cases as higher depth has a direct correlation to increased runtime and gate induced noise.

# Chapter 5

## Conclusion

This thesis proposes an improved design of an quantum integer multiplier inspired by the classical array multiplier: the Quantum Array multiplier (QAM). The new approach leverages quantum mechanics by performing the addition through controlled rotations in the phase domain, while at the same time, it increases parallelism through array multiplication. This design avoids the need to transfer information from one row to the next in the array, and it also cancels the need to pass carry information, thanks to the accumulated rotation on each individual qubit of the resulting product. In addition to this, work in approximating operations in the phase domain was also incorporated in the form of the AQAM algorithm. This allowed for a significant reduction in depth without much loss in accuracy on noisy systems.

Another three quantum integer multipliers were implemented and compared against the new approach: one similar to a classical implementation, through repeated additions on the computational basis (CB); a second one which initially proposed the repeated addition through phase rotation (RA); and a third one which utilizes weighted additions to perform the multiplication without repetition (QFM). The width of all designs was shown to be equivalent, and not necessarily a limiting factor for these computations. The depth was also compared for each of the designs, as an approximate metric of scalability and performance. For small magnitude operands, repeated addition performs better than the new QAM, since the structure of QAM is always the same no matter how small the value of the operand, while the others (CB and RA) only repeat additions as many times as indicated by the multiplier. But for the cases tested, as soon as operands surpassed a specific magnitude, the depth was many times reduced in the case of QAM. In fact QAM's depth grows with polynomial $O(n^3)$, while the repeated addition algorithms display exponential growth in depth. In comparison to QFM, the QAM rises much slower, incurring only slightly over half the depth by a bit width of 8. AQAM improves this further by scaling as with just over a polynomial $O(n^2)$ trend.

Due to limited access to quantum hardware, implementations for real performance results and accuracy were not tested using actual quantum computers. Instead, simulations with and without noise show how these may function on said hardware. This paper shows the new QAM's potential as an efficient implementation an quantum integer multiplier, with the addition of the AQAM for when a lower depth is required and a slight loss in ideal accuracy is not detrimental. This will particularly benefit quantum algorithms that require multiplication of integers. By implementing the multiplication directly on the quantum hardware, costly quantum-classical data exchange is avoided.

# Bibliography

[1] Peter W. Shor. “Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer”. In: *SIAM J. Comput.* 26.5 (Oct. 1997), pp. 1484–1509. ISSN: 0097-5397. DOI: 10.1137/S0097539795293172. URL: http://dx.doi.org/10.1137/S0097539795293172.

[2] Lov K. Grover. *A fast quantum mechanical algorithm for database search.* arXiv:quant-ph/9605043. Nov. 1996. DOI: 10.48550/arXiv.quant-ph/9605043. URL: http://arxiv.org/abs/quant-ph/9605043 (visited on 10/28/2023).

[3] 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. DOI: 10.1109/SFCS.1994.365700.

[4] John Preskill. “Quantum Computing in the NISQ era and beyond”. In: *Quantum* 2 (Aug. 2018), p. 79. ISSN: 2521-327X. DOI: 10.22331/q-2018-08-06-79. URL: http://dx.doi.org/10.22331/q-2018-08-06-79.

[5] John A. Cortese and Timothy M. Braje. *Loading Classical Data into a Quantum Computer.* arXiv:1803.01958 [quant-ph]. Mar. 2018. DOI: 10.48550/arXiv.1803.01958. URL: http://arxiv.org/abs/1803.01958 (visited on 10/24/2023).

[6] G. Florio and D. Picca. *Quantum implementation of elementary arithmetic operations.* arXiv:quant-ph/0403048. Mar. 2004. DOI: 10.48550/arXiv.quant-ph/0403048. URL: http://arxiv.org/abs/quant-ph/0403048 (visited on 02/18/2023).

[7] Sashwat Anagolum. *Arithmetic on Quantum Computers: Multiplication.* en. May 2020. URL: https://medium.com/@sashwat.anagolum/arithmetic-on-quantum-computers-multiplication-4482cdc2d83b (visited on 02/16/2023).

[8] Lidia Ruiz-Perez and Juan Carlos Garcia-Escartin. “Quantum arithmetic with the Quantum Fourier Transform”. In: *Quantum Information Processing* 16.6 (June 2017). arXiv:1411.5949 [quant-ph], p. 152. ISSN: 1570-0755, 1573-1332. DOI: 10.1007/s11128-017-1603-1. URL: http://arxiv.org/abs/1411.5949 (visited on 09/11/2023).

[9] Isaac L. Chuang, Neil Gershenfeld, and Mark Kubinec. “Experimental Implementation of Fast Quantum Searching”. In: *Physical Review Letters* 80.15 (Apr. 13, 1998). Publisher: American Physical Society, pp. 3408–3411. DOI: 10.1103/PhysRevLett.80.3408. URL: https://link.aps.org/doi/10.1103/PhysRevLett.80.3408 (visited on 03/25/2024).

[10] Stephan Gulde et al. “Implementation of the Deutsch–Jozsa algorithm on an ion-trap quantum computer”. In: *Nature* 421.6918 (Jan. 2003). Publisher: Nature Publishing Group, pp. 48–50. ISSN: 1476-4687. DOI: 10.1038/nature01336. URL: https://www.nature.com/articles/nature01336 (visited on 03/25/2024).

[11] Frank Arute et al. “Quantum supremacy using a programmable superconducting processor”. en. In: *Nature* 574.7779 (Oct. 2019). Number: 7779 Publisher: Nature Publishing Group, pp. 505–510. ISSN: 1476-4687. DOI: 10.1038/s41586-019-1666-5. URL: https://www.nature.com/articles/s41586-019-1666-5 (visited on 10/31/2023).

[12] Yulin Wu et al. “Strong Quantum Computational Advantage Using a Superconducting Quantum Processor”. In: *Physical Review Letters* 127.18 (Oct. 2021). Publisher: American Physical Society, p. 180501. DOI: 10.1103/PhysRevLett.127.180501. URL: https://link.aps.org/doi/10.1103/PhysRevLett.127.180501 (visited on 10/31/2023).

[13] Nick S. Blunt et al. “Perspective on the Current State-of-the-Art of Quantum Computing for Drug Discovery Applications”. In: *Journal of Chemical Theory*

*and Computation* 18.12 (Dec. 13, 2022). Publisher: American Chemical Society, pp. 7001–7023. ISSN: 1549-9618. DOI: 10.1021/acs.jctc.2c00574. URL: https://doi.org/10.1021/acs.jctc.2c00574 (visited on 03/26/2024).

[14] Qiskit contributors. *Qiskit: An Open-source Framework for Quantum Computing.* 2023. DOI: 10.5281/zenodo.2573505.

[15] Craig Gidney. *Asymptotically Efficient Quantum Karatsuba Multiplication.* arXiv:1904.07356 [quant-ph]. Apr. 2019. DOI: 10.48550/arXiv.1904.07356. URL: http://arxiv.org/abs/1904.07356 (visited on 10/20/2023).

[16] Kevin Hartnett. *A New Approach to Multiplication Opens the Door to Better Quantum Computers.* en. Apr. 2019. URL: https://www.quantamagazine.org/a-new-approach-to-multiplication-opens-the-door-to-better-quantum-computers-20190424/ (visited on 10/20/2023).

[17] Mehdi Ramezani, Morteza Nikaeen, Farnaz Farman, Seyed Mahmoud Ashrafi, and Alireza Bahrampour. *Quantum Multiplication Algorithm Based on Convolution Theorem.* arXiv:2306.08473 [quant-ph]. June 2023. DOI: 10.48550/arXiv.2306.08473. URL: http://arxiv.org/abs/2306.08473 (visited on 10/22/2023).

[18] V. Vedral, A. Barenco, and A. Ekert. "Quantum Networks for Elementary Arithmetic Operations". In: *Physical Review A* 54.1 (July 1996). arXiv:quant-ph/9511018, pp. 147–153. ISSN: 1050-2947, 1094-1622. DOI: 10.1103/PhysRevA.54.147. URL: http://arxiv.org/abs/quant-ph/9511018 (visited on 08/30/2023).

[19] Thomas G. Draper. *Addition on a Quantum Computer.* arXiv:quant-ph/0008033. Aug. 2000. DOI: 10.48550/arXiv.quant-ph/0008033. URL: http://arxiv.org/abs/quant-ph/0008033 (visited on 03/20/2023).

[20] Adriano Barenco, Artur Ekert, Kalle-Antti Suominen, and Päivi Törmä. “Approximate Quantum Fourier Transform and Decoherence”. In: *Physical Review A* 54.1 (July 1996). arXiv:quant-ph/9601018, pp. 139–146. ISSN: 1050-2947, 1094-1622. DOI: `10.1103/PhysRevA.54.139`. URL: `http://arxiv.org/abs/quant-ph/9601018` (visited on 09/29/2023).