Quantum Array Multiplier

Aden Crimmins*, Sonia Lopez Alarcon† and Matthew Klein, Matthew Krebs and Sagar Kate
KGCOE Department of Computer Engineering, Rochester Institute of Technology
Rochester, New York
Email: *abc8255@rit.edu, †slaeec@rit.edu

2023 IEEE International Conference on Rebooting Computing (ICRC) | 979-8-3503-8204-4/23/$31.00 ©2023 IEEE | DOI: 10.1109/ICRC60800.2023.10386449

***Abstract*—This paper proposes a new and improved implementation of a quantum integer multiplier. Performing arithmetic computations is sometimes a necessary step in the implementation of quantum algorithms. In this work, Quantum Fourier Transform is used in order to perform scalable arithmetic in a generic bit-width quantum system. In the phase domain, addition can be implemented through accumulated controlled rotations on the qubits' state. Leveraging this, and inspired by the classical implementation of an array multiplier, a new integer multiplier is fully designed and tested in a quantum environment.**

The depth of a quantum circuit is the number of computational steps necessary to completion, and it is a key parameter that reflects on the performance of the design. The new design reduces the quantum depth of the design from the exponential order of the previously proposed designs to polynomial order.

***Index Terms*—Quantum Computing, Quantum Multiplication, Quantum Fourier Transform, Array Multiplier**

I. Introduction

THE subject of Quantum Computing is in its early stages of development, and new technologies and algorithms are constantly evolving. Much like classical computing, small-scale circuit designs are used as building blocks for large-scale algorithms. One such example described by Sashwat Anagolum pursues scaling the arithmetic of existing quantum adder designs to develop a quantum multiplier [1].

The importance of an efficient quantum multiplication algorithm stems from the wide range of applications that benefit from this operation. Improvements made to a quantum multiplication algorithm could prove useful for different applications, such as modular multiplication in Shor's algorithm [2], or could play a role in the implementation of oracles for Grover's search algorithm [3].

Designing principle circuits through quantum means is usually done in the hope of achieving a computational speedup in comparison with classical computation. On the other hand, encoding data in a quantum state and reading it out is computationally expensive, and hybrid quantum-classical implementations require data transfers that hurt performance. Hence, when a basic computation cannot be implemented with better performance in its quantum version, it may still be beneficial for larger quantum applications that would have to be hybrid otherwise, exchanging information back and forth between the quantum and classical worlds.

The design previously proposed [1] takes in integers as binary inputs and computes their product as a binary output. This implementation arrives at a product through a quantum circuit that takes advantage of the Quantum Fourier Transform (QFT) and the phase domain.

In the Noisy Intermediate-Scale Quantum (NISQ) era [4], noise is a critically limiting factor. In addition, computations are constantly challenged by decoherence time, the time that it takes for a quantum system to 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. As technology improves and the number of qubits available grows while the error and noise levels decrease, reaching the post-NISQ era, the depth of the circuit will still be a critical factor that needs to be reduced, with a direct impact on the performance of the computation. Depth is a metric of performance and the parallelism of the computation.

This paper looks at the scalability of the implementations and takes a close look at the depth of the designs for future implementations. As this paper will show, the depth of the computation of the original phase basis design scales exponentially with the size in classical bits of the operands. To achieve improved performance results, and given the sensitivity to noise and error rates of current quantum computers, it is critical to reduce, to the extent possible, the depth of the computation.

The scalability problem of quantum circuit design is ubiquitous in this field, but it is exacerbated by naive design practices. A good understanding of the properties of quantum algorithms leads to more efficient designs with reduced requirements for the number of qubits and quantum gates. On the other hand, the description of quantum applications and circuits displays 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.

This paper's design offers efficiency improvement over existing quantum multipliers coming from two sources: the efficient use of addition on the phase domain (a quantum property) and the classical hardware design of array multipliers. The depth of a quantum circuit is the number of computational steps necessary to completion, and it is a key parameter that

This material is based upon work supported by the National Science Foundation under Award No. 2300476

reflects on the performance of the design. Reducing depth is based on the efficiency of the computations as well as how parallelism is exploited through them.

Section II briefly reviews significant work in the field of quantum arithmetic to put this paper into context. Section III describes the original phase basis multiplier, some immediate improvements to it, and the new proposed approach. Results comparing these designs are presented in Section IV, followed by the key takeaways in Section V.

II. Background and Related Work

Quantum arithmetic has been developing for a few decades, with new algorithms and techniques being proposed for efficient and fast operations on qubits. Primarily, the algorithms developed were only for integer arithmetic; over time, the operations on the floating point were also considered.

Adders are relevant to this discussion, as multipliers always involve additions in one way or another. The implementation of quantum addition using a full adder is given precisely by Sohel et al. [5]. A basic full adder can be described with CNOT and Toffoli quantum gates. The n-qubit design requires an n-qubit quantum register and an n-bit classical register to hold the values for input and output in the quantum domain and the measured value in the classical domain. The paper demonstrated the complete implementation of the full adder truth table, with multiple intermediate steps in which the qubits are used to store the intermediate values that are not required at the end. The ripple carry adder circuit proposed by Vedral et al. [6] demonstrates the addition of two n-bit numbers as input and outputs a single bit, the high bit of the sum. These circuits require $n-1$ ancillae qubits, with $4n+1$ CNOT gates and $4n+1$ Toffoli (doubly controlled-NOT) gates, resulting in limited parallelism. The circuit was improved and was reduced to one ancilla qubit proposed by Cuccaro et al. [7] with $2n+1$ Toffoli gates, $5n+O(1)$ CNOT gates, and $2n+1$ negations; the estimated depth is $2n+1$.

Floating point arithmetic falls outside of the scope of this paper. The paper by Seidel et al. [8] on Efficient Floating Point Arithmetic for Quantum Computers proposed encoding the mantissa and the exponent part separately as quantum variables, known as bi-quantum encoding. Depending on the state of the exponent register, the mantissa would need to be bit shifted to add two bi-quantum encoded floats. Theoretically, a circuit like this could be built using Fredkin gates and an incrementor gate, but it would be exceedingly complicated. An additional challenge of floating point computations is the encoding of quantum information. The encoding of the input as quantum states is challenging; the opposite, where the result that is in a quantum state is converted to classical data, is the output problem. In general, these two issues are difficult to tackle effectively in a quantum computer, and often it is assumed that the data is available in quantum states by some means. Another crucial concept is studying quantum algorithms that include all-quantum input and output data. Examples include quantum machine learning for quantum data, principal component analysis, quantum simulators, etc. The proposed solutions assume an efficient quantum algorithm exists to achieve the quantum state preparation of the rows and columns of a matrix, such as by QRAM [9].

Due to the unique characteristics of quantum information and the core linear algebra features of quantum mechanics, some linear algebra computations may be more immediate to implement than bit-wise computations. Matrix multiplication is one of these computations. The paper on matrix multiplication [10] discusses the use of Swap test [11], SVE [12] and HHL [13] algorithms for quantum matrix multiplication and the complexity of preparing the required quantum states. The HHL algorithm requires the input matrix to be Hermitian and the vector to be in amplitude encoding. The preparation of these steps can take significant computational resources.

The focus of this paper is on the efficient multiplication of two integers in a quantum environment. Some quantum algorithms may require a multiplication of two integers as a necessary step, and as stated above, the transfer of information back and forth between the classical and quantum environments may be prohibitive in terms of time and resources. A quantum implementation of a multiplier of two integers was proposed in [1] to make use of repetitive addition of a multiplicand, a number of times determined by the value of the multiplier. This approach to the multiplication of integers is commonly utilized by both people and computers. The simplicity behind this idea makes it easy to implement and scale to fit one's needs. The paper takes this as the starting point of the new proposed design.

A integer multiplier architecture can also be found in [14]. In this work, like that by Anagolum [15], the proposed QFT and its inverse (IQFT) are unnecessarily repeated, as will be shown. When it comes to the multiplier of integers, the work by Ruiz-Perez and Garcia-Escartin [16] proposes a similar approach to the one discussed here. The mathematical description is described, however, does not include any implementations, and like the previous two [15], [14] it does not discuss or compare the depth of the design.

III. Quantum multiplier designs

The original quantum multiplier design in the phase basis [1] is based on the repeated addition of the multiplicand in an accumulator, as many times as the multiplier indicates. This repeated addition takes place in the phase domain. The basic functionality of this multiplier is as follows: The initial state of the accumulator is encoded in the phase domain through a QFT, so the multiplicand can control the rotations of the qubits' phases depending on the significance of the qubit upon which the rotation is acting. The multiplier is then decremented by 1 (the value stored in the ancillary qubit), and this cycle is repeated until the multiplier has reached 0. Last, the accumulator is returned to encoding in the computational basis through an IQFT to extract the result of the multiplication.

Therefore, the design uses four main groups of qubits: (1) the multiplier, (2) the multiplicand, (3) the accumulator, and (4) one ancillary qubit. Figure 1 represents these four groups and their initialization stage in the case of a 1x1 multiplier. According to this, the width of the system is given by $2(n+$

$m) + 1$, where n represents the number of multiplier qubits and m represents the number of multiplicand qubits.

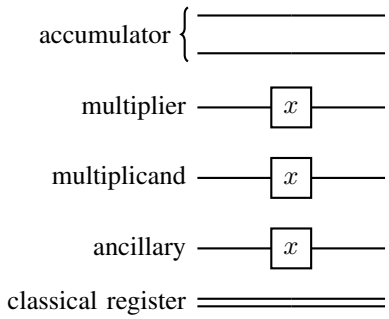

Fig. 1: Initialization for the multiplier architecture, implementing 1x1 in this case. The accumulator by default is initialized to $|00\rangle$, while multiplier and multiplicand both take states $|1\rangle$ and $|1\rangle$ to represent the 1's in the multiplication. One ancillary qubit is necessary to implement the repeated subtraction of 1 to decrement the multiplier.

The implementation is composed of three elementary building blocks: the QFT, the evolve stage, and the IQFT.

A. QFT and IQFT

In order to place the 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 1.

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

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 2. Table I 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.

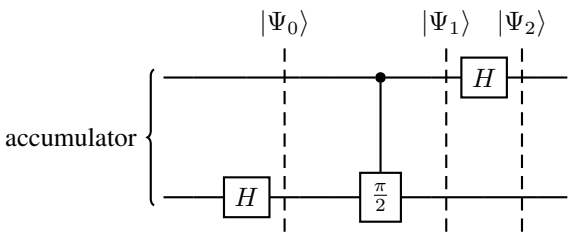

Fig. 2: Quantum fourier transform stage.

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. The IQFT is performed as described by Figure 2. 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 QFT, which contributes to the rapid growth in depth as bit width increases.

$$QFT = \frac{1}{\sqrt{N}} \sum_{j,k=0}^{N-1} e^{\frac{-i2\pi jk}{N}} |j\rangle\langle k| \tag{2}$$

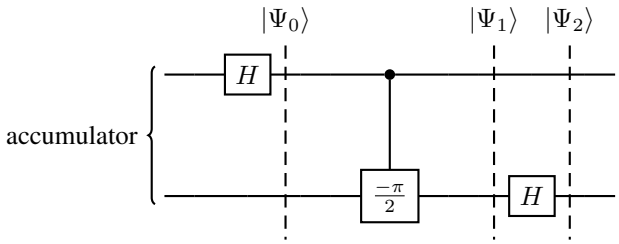

Fig. 3: Inverse quantum fourier transform stage.

B. A two qubit example of addition in the phase domain.

Table I [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 $|01\rangle$, π 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 π 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:

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

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.

Looking at Table I, it can be noticed that adding two numbers can be achieved by 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

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

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

TABLE I: QFT breakdown 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 QFT 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	$\|11\rangle$	$\|00\rangle - i\|01\rangle - \|10\rangle + i\|11\rangle$	$(\|0\rangle - \|1\rangle)(\|0\rangle - i\|1\rangle)$

on each result qubit are controlled by each of the addend's qubits.

With this in mind, a multiplication can be performed through the accumulation of the multiplicand a number of times indicated by the multiplier using accumulated rotations in the phase space. This evolution of the QFT state towards the accumulated QFT is implemented in the *evolve stage*, described next.

C. Evolve Stage

The evolve stage is responsible for the addition acting upon the accumulator by the multiplicand, which is repeated each iteration. The evolve stages occur after the accumulator has been placed into the phase domain. Figure 4 depicts the evolve stage of a one-qubit accumulation on a two-qubit register through rotations on the most and least significant qubits.

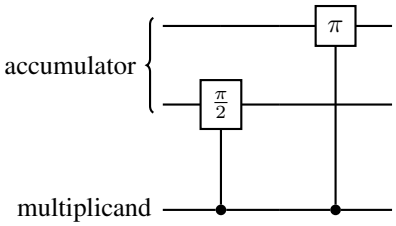

Fig. 4: Evolve stage of a one qubit multiplicand on a two qubit accumulator. π and $\frac{\pi}{2}$ controlled rotations are employed upon the most and least significant bits of the accumulator, using the multiplicand qubit as the control.

This addition could have also been done entirely within the computational basis using a combination of C-NOT and Toffoli gates, acting as classical XOR and NAND gates respectively, [17]. However, as it will be shown, the depth of the computational basis chained full adder circuit, especially when the size of the multiplicand is large, would be increasingly inefficient compared to a series of phase rotations.

D. Subtractor

The purpose of this stage is to decrement the multiplier by a constant value of "1" and then check if the multiplier is equal to "0" via a classical measurement. If so, the algorithm stops. If the multiplier is non-zero, another round of the algorithm must be performed.

The subtractor stage of the original multiplication algorithm looks very similar to the addition stage. This is because it utilizes the same theory and stages, but in this case, it is acting on the multiplier. By inverting the phase of the controlled rotation gates that were performed within the addition algorithm, subtraction is performed rather than addition.

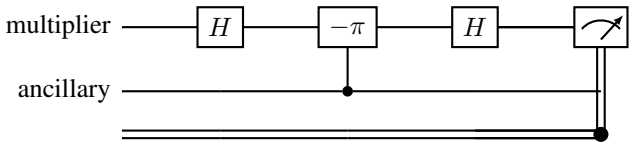

Fig. 5: Decrement of Multiplier - Combination of the QFT, IQFT, and Evolve Stages.

A measurement of the multiplier qubits is performed every iteration. This measurement can be done without collapsing the quantum state of the accumulator or the multiplicand because the multiplier qubits have no entanglement with the other qubits within the circuit.

E. Improvements on the original design

Two primary improvements were made to the original design: (1) the removal of the ancillary bit used in the subtraction, and (2) the removal of iterative QFT and IQFT.

1) Removal of the ancillary qubit: This was done by making a custom *sub1* method that does not require a second argument to use as a control. Because there is always a subtraction of one from the multiplier, there is no reason to use an ancillary bit to store the value. Instead, an uncontrolled phase rotation of the bits within the multiplier can be directly applied. For the scenario of 1x1, this is simply a single phase gate with -π rotation. Figure 6 shows the improvement that removes the need for the controlled-rotation gate (Figure 5). This only reduces the total qubit count by one, but in this scenario, every qubit counts.

2) Removing the iterative QFT and IQFT: The original design transformed the accumulator in and out of the phase domain with each iteration, with QFT-IQFT implementations. This design can be improved. At the beginning of the algorithm, the accumulator is placed into the quantum phase domain via the QFT, then the evolve stage is run as many times as needed, and finally, only at the end does the accumulator transform back into the computational domain via the IQFT so that it can be measured. Doing this removes the unnecessary transformations into and out of the phase domain and does not affect the algorithm's functionality because the multiplier has no entanglement with the accumulator. Therefore, each iteration, it can be measured without affecting the quantum state of the accumulator qubits. Figure 6 illustrates the complete multiplication of $1x1$ after the improvements are implemented. The original proposed design intended to apply IQFT after each accumulation and

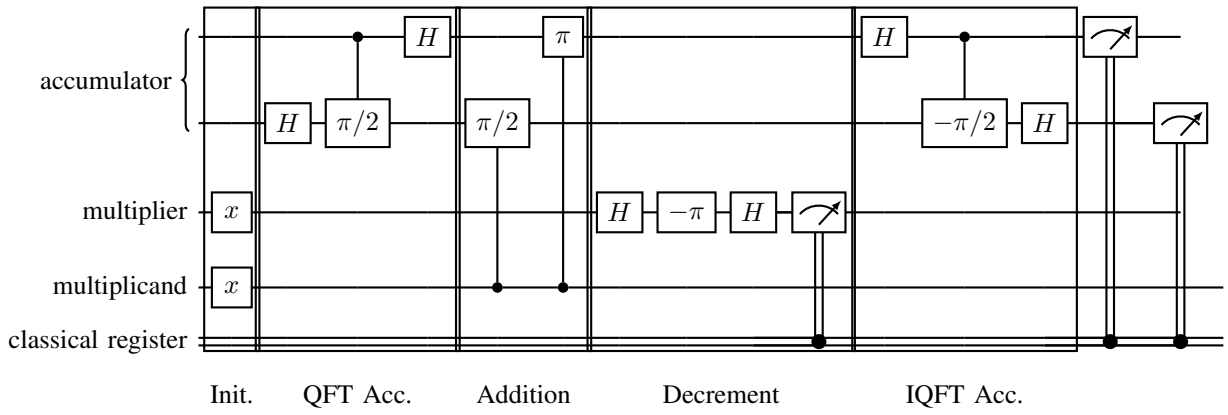

Fig. 6: Full 1x1 Multiplication Circuit Post Improvements. Each section is outlined to improve readability.

QFT again before the next iteration. This improvement upon the original proposal [1] significantly reduces the depth of the accumulation process, as will be shown in Section IV.

F. New quantum array multiplier approach

The array multiplier approach has been used in the design of classical multipliers and leverages parallelism for better performance [18] in different contexts, and in particular for Field Programmable Gate Array (FPGA) hardware design, and as such, it has proven its effectiveness. The array multiplier is a combinational circuit that, in the classical domain, multiplies two binary numbers using a shifted array structure, as shown in Figure 7.

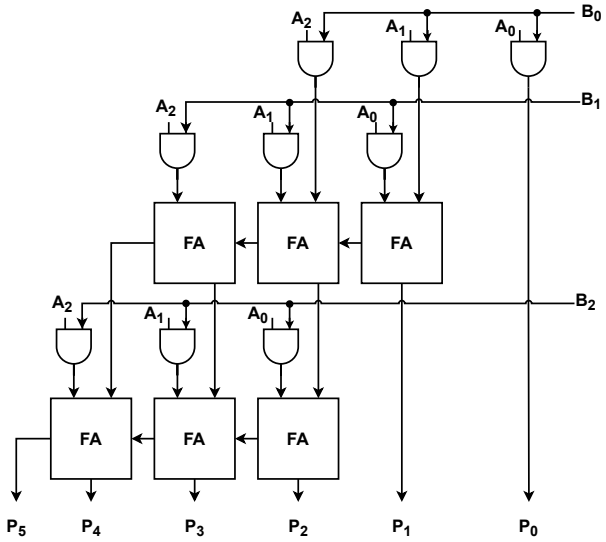

Fig. 7: Structure of an Array Multiplier architecture utilizing both Full Adders and AND logic gates.

The basic building block of an array multiplier is a single-bit Full Adder (see Figure 8). The function of a full adder is to add two single-bit binary values, a and b, along with a carry-in, c_i, outputting a one-bit addition, Sum, and a carry-out, c_0. Carries are necessary in the case of overflow (i.e. binary $1 + 1 = 10$).

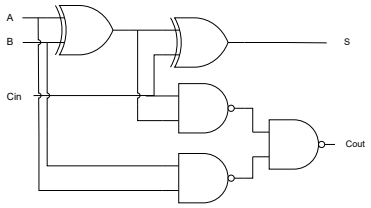

Fig. 8: Classical full adder logic circuit design.

Let the multiplier be A (bits a_{n-1}-a_0) and the multiplicand be B (bits b_{m-1}-b_0). Similarly to the pencil and paper algorithm learned at school, every pair of bits a_i, b_j, needs to be multiplied, shifted, and added together in columns. Therefore, when multiple Full Adders are strung together in a single row, it allows for a multi-bit addition to take place, in which the individual bit addition occurs along with any carries that would normally take place in full addition. In order to create a multiplier using these full adders, the addition takes place after each pair of bits has been multiplied (by AND gates), and these need to be connected in both rows and columns as shown in Figure 7. The number of columns depends on the bit size of input A, while the number of rows is determined by the number of bits in B. Each row following the first is shifted once to the left to signify an increase in the magnitude of the resulting sum of the row. The resulting product has a bit width equal to the sum of the input bit widths. Using this architecture, regardless of input values, an output product is guaranteed to be correct and without overflow.

The classical design is very efficient on hardware because it is easily parallelized. For an nxn bit multiplier circuit, only n rounds of addition need to be performed. Hence, the array multiplier has an $O(n)$ complexity, whereas the non-parallel repeated addition multiplier has an O(2^n) time complexity. The key concept here, pertaining to the proposed implementation, is the structure composed of AND gates and one-bit full adders.

This concept is applicable to the phase domain quantum multiplier. Figure 11 illustrates the example of a two-qubit

multiplier ($p = a * b$). The product will be placed in the phase domain through a QFT. A *double control rotation* takes the place of each and-gate pair *and* the corresponding addition if the shape of a rotation in the phase domain. These rotations are implementing the additions in the same way as they were explained in Section III-B, and Table I.

In the classical design, each full adder takes in both the result of the current row and the previous row. In the quantum design, it is not necessary to have any dependencies on the previous row due to the fact that each row provides the phase shift directly to the product. The accumulation of the rotations on each of the product qubits also takes care of the carry dependencies that do not exist in the quantum implementation. This allows the product of the two inputs to be successfully multiplied. There is no need to decrement any multipiers, measure for comparison against zero, or repeat QFT-IQFT blocks; only one is required at the beginning and end of the computation, acting on the product quantum array.

It should be noted that one drawback of this circuit design is that, while it does not need to measure qubits throughout the computation, it does not reduce in size when provided with inputs that are smaller than the bit size. In the repeated addition, the addition only needed to be performed as many times as the multiplier indicated. In the new Quantum Array Multiplier (QAM), the structure is generic, and only the encoding changes the operands and product. Therefore, the depth of the circuit depends on the size of the operands and not on their values. As we will show in Section IV, this drawback is quickly overcome by a much more efficient design.

G. Computational Basis Multiplier: one more multiplier for comparison

The use of the QFT was prompted by the poor scalability of the strictly computational basis implementation. This implementation is based on repeated addition, but this addition is performed in the computational basis. In order to perform the same operation while staying in the computational domain, multiple Toffoli gates were used, as shown in Figure 12.

IV. Results

With these proposed improvements and new approach, quantum circuits were implemented and simulated without noise on the Qiskit Aer Simulator [19]. The results were always correct in all cases. As per performance, since the execution was not performed on real hardware, we present the depth 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.

A. Summary of designs

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

- Original Phase Basis (OPB): 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 multipler is decremented towards a zero value that indicates the end of the computation. This addition is performed in the Phase domain by applying QFT and IQFT every time the accumulator is incremented. This implementation also requires the measurement and comparison of the multiplier.
- Improved OPB (I_OPB): This design follows the same approach as OPB. In this case, however, one single QFT and IQFT stage is needed on the accumulator. In addition, the subtraction on the multiplier does not require an ancilla qubit to indicate that the value 1 is being subtracted. QFT and IQFT remain on the multiplier for each increment of the accumulator.
- Computational Basis Multiplier (CBM): for comparison, the repeated addition is also performed on the computational basis as opposed to the phase domain, through the use of Toffoli and CNOT gates.
- Quantum Array Multiplier (QAM): for the new approach inspired by the classical array multiplier, double-controlled rotations realize bit multiplication and consequent addition.

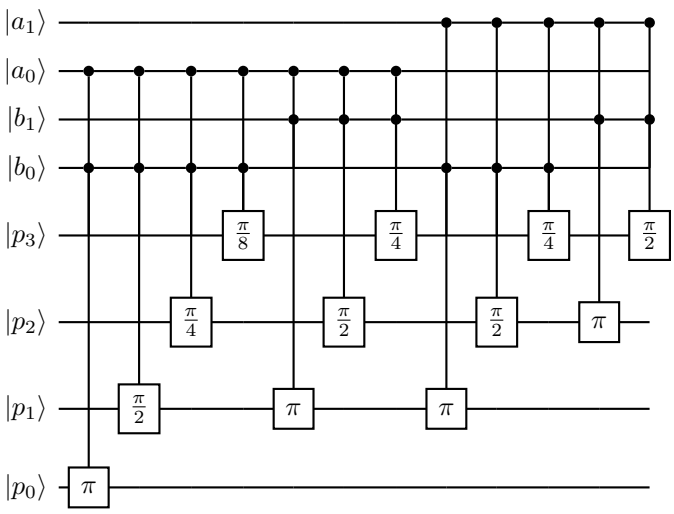

Fig. 9: Quantum multiplier of 2-qubit operands. The product $|p\rangle$ register is already placed in the phase domain after its QFT block (not shown in this figure). Rotations are applied to the product when the multiplication of the two qubits results in a 1-value to be added to each corresponding product qubit in the phase domain.

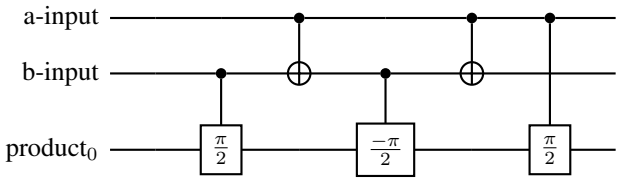

Fig. 10: Double control rotation decomposition

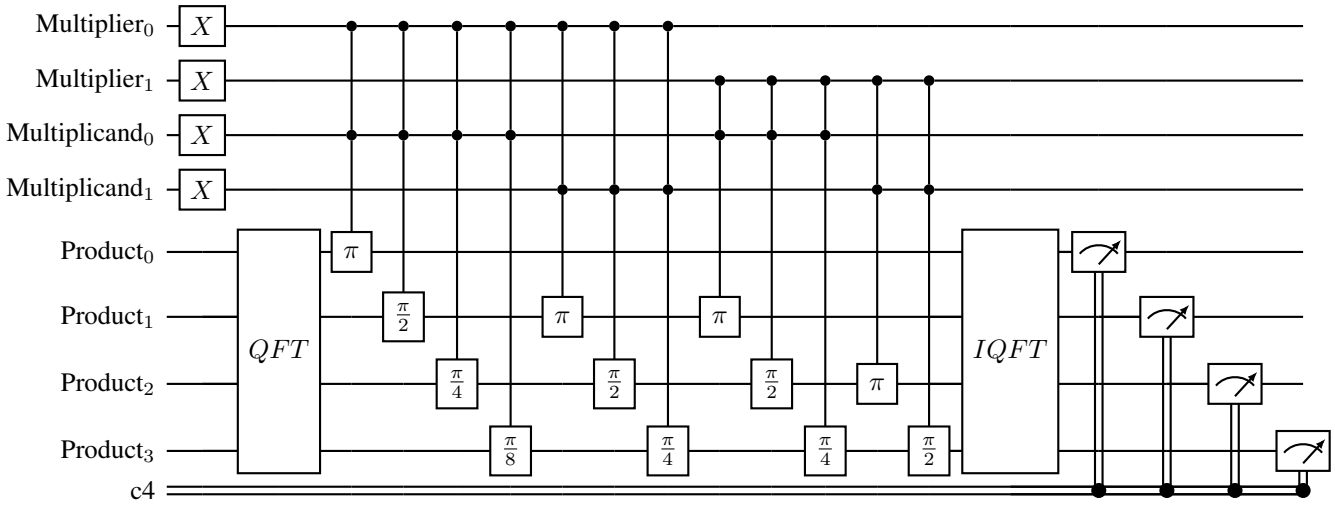

Fig. 11: Quantum Array Multiplier (QAM) with each stage connected starting with the QFT stage, followed by each of the row additions and finally the IQFT.

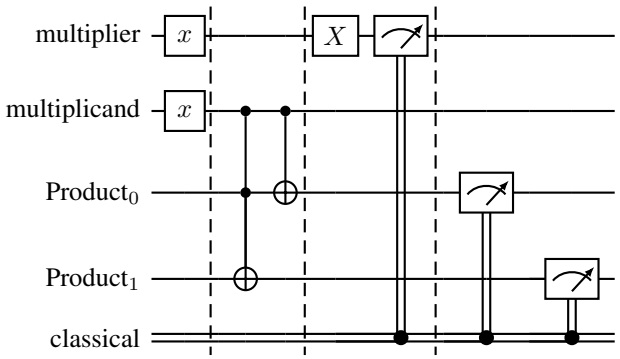

Fig. 12: Computational basis multiplication of two 1-qubit numbers. The result is stored in the output qubits, which is then measured into the classical register.

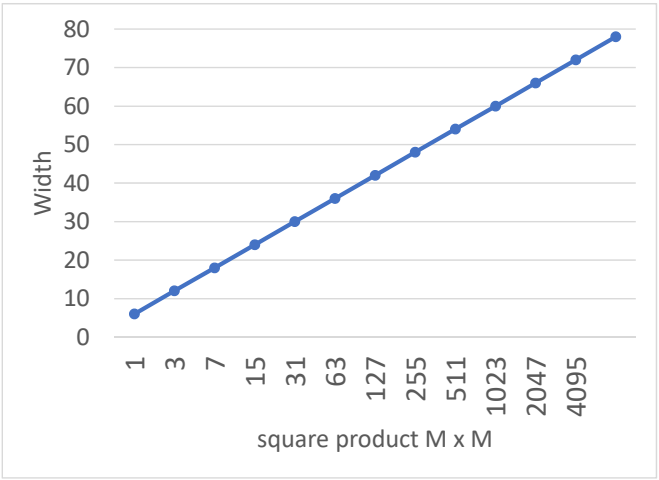

Fig. 13: Width for all compared circuits, for square products of up to 12 qubits per operand (maximum operand decimal value is 4095 for the 12 qubit case). Variations of +/- 1 qubit occur across designs, but all designs follow the same trend.

B. Depth and width of the compared circuits

1) Width of quantum multipliers: All four compared quantum multipliers use almost the same number of qubits for the same size operands. The number varies by only one more qubit when the computation takes place in the Original Phase basis implementation. Therefore, the width is $width = 3(m * n)$ where m and n are the size in bits (qubits) of the two integer operands. Figure 13 plots the number of qubits used in the implementation of square products of MxM numbers (up to 12 qubits per operand). According to this plot and trend, the number of qubits is not necessarily a limiting factor for the implementation of the integer multiplier, and in any case, it is not a factor that can be improved across the proposed designs since it remains equal or very similar for all of them.

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 partially depends on the parallelism that the computation is able to extract out of the gates acting on its corresponding qubits.

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 11, 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 III, the original phase-basis multiplier, based on repeated addition, 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.

Figure 14 plots the Qiskit reported depth for the trivial product Mx1. Since OPB (Original Phase-Basis) only has to add onto the accumulator once, while the QAM (Quantum Array Multiplier) performs all the double-controlled rotations for the general MxM multiplier circuit, OPB depth is significantly lower for the cases shown, up to 4095x1 (12 qubits per operand). The trend continues as shown for this trivial case, and there is no difference in depth for the Phases basis multipliers, OPB and I_OPB (improved) cases. The computational basis multiplier is slightly better than the OPB or I_OPB cases up to the $M = 127$ case. After that, CBM's circuit is deeper than the OPB cases. As it will be shown, CBM scales worse than the rest of them for trivial or square products.

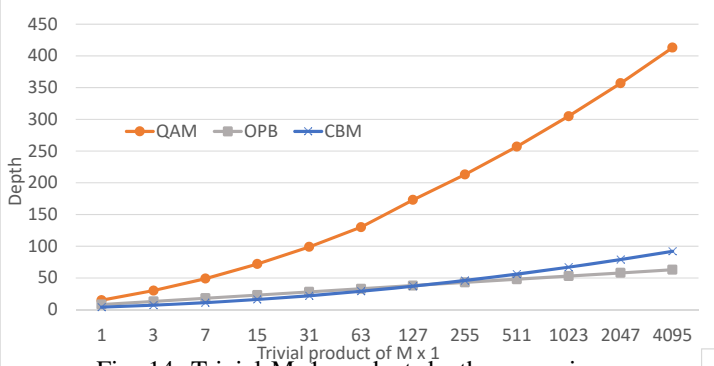

Fig. 14: Trivial Mx1 product depth comparison.

Figure 15 plots the depth for the other extreme case, the MxM square product. These numbers are reported for operands of up to 12 qubits (4095 decimal number) for all designs listed at the beginning of this section. Two lines are plotted for each of these designs: (1) the Qiskit reported depth value for each design and product size, and (2) the calculated trend for each of them. The trends are only approximate but fit exponential trends for the original, improved, and computational designs, while the new proposed Quantum Array Multiplier shows a polynomial $O(n^3)$ trend.

The behavior of the new QAM design is so much better than the other three that it is hard to appreciate its depth in Figure 15. Table II shows the specific depth number for each design 12 qubit case (4095) case. The computational basis multiplier had a depth of over 864K and was not plotted in Figure 15. To better appreciate the depth differences between the designs,

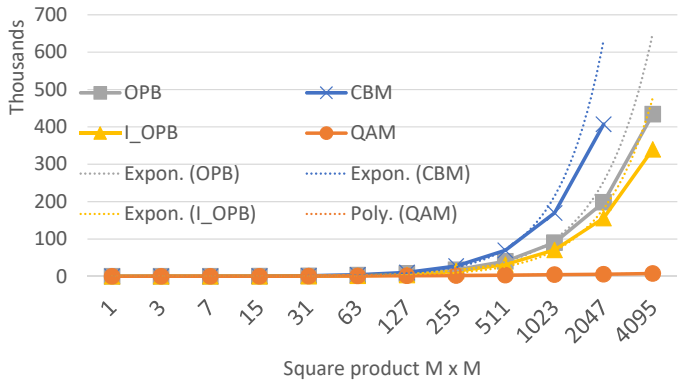

Fig. 15: Comparison of reported depth (in thousands) and calculated trends for the square MxM multiplier designs.

Figures 16 and 17 display the relative depth of the original and improved designs (CBM, OPB, and I_OPB) against the new proposed QAM design.

Design	Depth	Relative Depth over QAM (-x better)
CBM	864058	114x
QPB	434071	57x
I_QPB	339981	45x
QAM	7561	1x

TABLE II: Reported depth for the 12 qubit, MxM case (4095x4095 in decimal value). The third column shows the improvement of QAM over the corresponding designs. 1x is QAM against itself.

Figure 16 displays the relative depth of each design compared to the new proposed QAM. For example, for the 4095 operand case, the depth of the QAM is over 114x better than the computational basis implementation (CBM), over 57x better than the original computation on the phase domain (OPB), and over 44x better than the improved phase domain implementation (I_OPB).

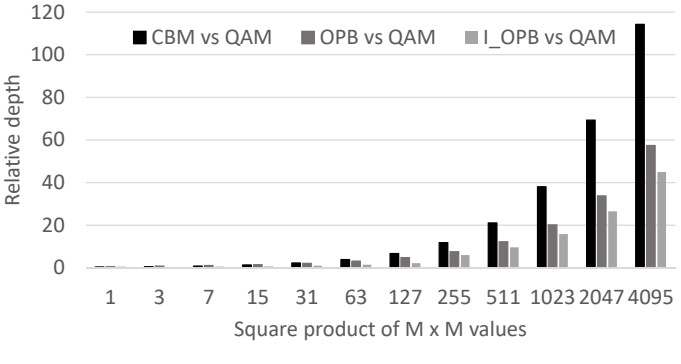

Fig. 16: Relative depth against the lowest depth case, QAM, for MxM cases ranging from 1 qubit operand to 12 qubit operand (decimal value 4095).

Figure 17 takes a closer look at the small size cases. Depth improvement thanks to QAM is shown when the bars surpass

the 1x line. Due to the fact that repeated additions will have a lower depth for smaller numbers, in the small operand cases, these bars do not reach the 1x line, showing that the fixed size multiplier implemented in QAM does have a higher depth than the others. However, QAM quickly improves, surpassing the 1x line for the 3 qubit case (operand $M = 7$) in the case of OPB, 4 qubits (operand $M = 15$) for CBM, and 5 qubits (operand $M = 31$) for I_OPB.

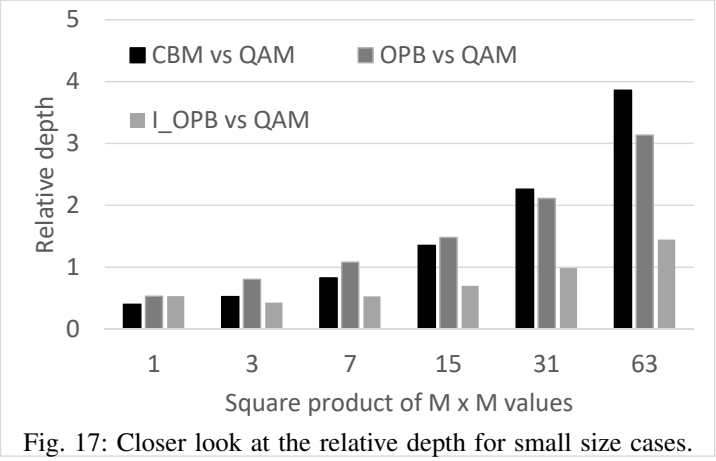

Fig. 17: Closer look at the relative depth for small size cases. Better depth of QAM can be seen in bars above the 1x line.

V. Conclusion

This paper proposes a new design of a 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 increasing 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.

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 (CBM); a second one, which initially proposed the repeated addition through phase rotation (OPB); and a third one with improvements upon the latter (I_OPB). The width of all designs was shown to be almost equivalent and not necessarily a limiting factor for these computations. The depth was also compared for all of them 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 operand, while the others (CBM, OPB, and I_OPB) only repeat additions as many times as indicated by the multiplier. But for the cases tested, as soon as operands surpassed magnitudes 7-31, the depth was many times reduced in the case of QAM. In fact, QAM's depth grows with polynomial $O(n^3)$, while the other algorithms display exponential growth in depth.

Due to limited access to quantum hardware, implementations for real performance results and accuracy were not tested. However, this paper shows the new QAM's potential as an efficient implementation of a quantum integer multiplier. This will particularly benefit quantum algorithms that require multiplication of integers. By implementing multiplication directly on the quantum hardware, costly quantum-classical data exchange is avoided.

References

[1] S. Anagolum. Arithmetic on quantum computers: Multiplication. [Online]. Available: https://medium.com/@sashwat.anagolum/arithmetic-on-quantum-computers-multiplication-4482cdc2d83b

[2] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," *SIAM J. Comput.*, vol. 26, no. 5, pp. 1484–1509, Oct. 1997. [Online]. Available: http://dx.doi.org/10.1137/S0097539795293172

[3] L. K. Grover, "A fast quantum mechanical algorithm for database search," in *Annual ACM Symposium on Theory of Computing*. ACM, 1996, pp. 212–219.

[4] J. Preskill, "Quantum Computing in the NISQ era and beyond," *Quantum*, vol. 2, p. 79, Aug 2018. [Online]. Available: http://dx.doi.org/10.22331/q-2018-08-06-79

[5] M. Sohel, N. Zia, M. Ali, and N. Zia, "Quantum computing based implementation of full adder," 11 2020, pp. 1–4.

[6] V. Vedral, A. Barenco, and A. Ekert, "Quantum networks for elementary arithmetic operations," *Phys. Rev. A*, vol. 54, pp. 147–153, Jul 1996. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevA.54.147

[7] S. A. Cuccaro, T. G. Draper, S. A. Kutin, and D. P. Moulton, "A new quantum ripple-carry addition circuit," 2004. [Online]. Available: https://arxiv.org/abs/quant-ph/0410184

[8] R. Seidel, N. Tcholtchev, S. Bock, C. K.-U. Becker, and M. Hauswirth, "Efficient floating point arithmetic for quantum computers," 2021. [Online]. Available: https://arxiv.org/abs/2112.10537

[9] V. Giovannetti, S. Lloyd, and L. Maccone, "Quantum random access memory," *Phys. Rev. Lett.*, vol. 100, p. 160501, Apr 2008. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevLett.100.160501

[10] C. Shao, "Quantum algorithms to matrix multiplication," 2018. [Online]. Available: https://arxiv.org/abs/1803.01601

[11] D. Gottesman and I. Chuang, "Quantum digital signatures," 2001. [Online]. Available: https://arxiv.org/abs/quant-ph/0105032

[12] I. Kerenidis and A. Prakash, "Quantum recommendation systems," 2016. [Online]. Available: https://arxiv.org/abs/1603.08675

[13] A. W. Harrow, A. Hassidim, and S. Lloyd, "Quantum Algorithm for Linear Systems of Equations," *Physical Review Letters*, vol. 103, no. 15, Oct 2009.

[14] A. Nguyen, "TR-2004007: Optimal reversible quantum circuit for multiplication." [Online]. Available: https://academicworks.cuny.edu/gc_cs_tr/243

[15] S. Anagolum. Arithmetic on quantum computers: Addition, faster. [Online]. Available: https://medium.com/@sashwat.anagolum/qftaddition-ce0a0b2bc4f4

[16] L. Ruiz-Perez and J. C. Garcia-Escartin, "Quantum arithmetic with the quantum fourier transform," vol. 16, no. 6, p. 152. [Online]. Available: http://arxiv.org/abs/1411.5949

[17] A. Muthukrishnan, "Classical and quantum logic gates." [Online]. Available: http://www2.optics.rochester.edu/~stroud/presentations/muthukrishnan991/LogicGates.pdf

[18] S. Brown and Z. Vranesic, "Fundamentals of digital logic with VHDL design," in *Fundamentals of digital logic with VHDL design*, 3rd ed. McGraw-Hill, pp. 291–295.

[19] Qiskit. Qiskit aer: A high performance simulator framework for quantum circuits. [Online]. Available: https://qiskit.org/aer