Optimizing Quantum Circuits for Modular Exponentiation

Rakesh Das
Department of Information Technology
Indian Institute of Engineering Science and Technology, Shibpur
India-711103
rakesh.rs2017@it.iiests.ac.in

Anupam Chattopadhyay
School of Computer Science and Engineering
Nanyang Technological University
Singapore-639798
anupam@ntu.edu.sg

Hafizur Rahaman
Department of Information Technology
Indian Institute of Engineering Science and Technology, Shibpur
India-711103
rahaman_h@it.iiests.ac.in

***Abstract*—Today's rapid progress in the physical implementation of quantum computers demands scalable synthesis methods to map practical logic designs to quantum architectures. There exist many quantum algorithms which use classical functions with superposition of states. Motivated by recent trends, in this paper, we show the design of quantum circuit to perform modular exponentiation functions using two different approaches. In the design phase, first we generate quantum circuit from a verilog implementation of exponentiation functions using synthesis tools and then apply two different Quantum Error Correction techniques. Finally the circuit is further optimized using the Linear Nearest Neighbor (LNN) Property. We demonstrate the effectiveness of our approach by generating a set of networks for the reversible modular exponentiation function for a set of input values. At the end of the work, we have summarized the obtained results, where a cost analysis over our developed approaches has been made. Experimental results show that depending on the choice of different QECC methods the performance figures can vary by up to 11%, 10%, 8% in T-count, number of qubits, number of gates respectively.**

***Index Terms*—Quantum Algorithm(QA), Modular Exponentiation, Quantum Error Correcting Codes (QECC), Linear Nearest Neighbor (LNN)**

I. INTRODUCTION

Quantum computers are expected to excel at certain computational tasks with an asymptotic advantage over their classical counterparts. Examples for such tasks include factoring [1] and the simulation of quantum chemical processes [2], [3]. quantum algorithms are known for their asymptotic behavior, exact runtime estimates are often lacking due to implementations for functions such as the one considered in this paper.

The main motivation for implementing this function for its usefulness in one of the most famous Shor's factoring algorithm. In [4], it shows the significance of modular exponentiation circuit, and we can perform a classical simulation of this circuit. Shor's factoring algorithm has mainly two main components (shown in Fig. 2),one is Modular Exponentiation, and other is *Quantum Fourier Transform* (QFT). In this paper, we will focus on the modular exponentiation as it is the most comprehensive part of the algorithm, and it is basically a classical circuit which is made up of classical gates. In addition to that, to address the issue of gate errors, we perform *Quantum Error Correction Codes* (QECC) on the circuits as well as we also consider *Linear Nearest Neighbor* (LNN) property.

Related Works: As there is a huge impact on the implementation details of some functions for the reality of a given quantum algorithm, there are several literature available which provide circuits for various low-level arithmetic functions such as addition[5], multiplication [6]. Furthermore, in [7] implementations of some higher-level arithmetic functions has been presented. Recently, [8] provides the reversible implementation of some arithmetic circuits for fixed-point representation where final circuits are set of Toffoli gates, whereas in [9] shows the implementation of an arithmetic circuit for floating point representation.

Similar works [10], [11] show quantum implementation of modular exponentiation circuit but without considering fault tolerance representation. For the practical realizable scalable quantum computer, it is necessary to make the quantum circuits protected from errors. There are many literature available which discuss the importance of fault-tolerant representation[12]. Another work [13] presents the fault tolerant implementation of quantum circuits of reciprocal function but here *Clifford+T* gate library is employed to construct fault-tolerant error protected quantum circuits. But as in terms of cost, *Clifford+T* gate library is quite expensive so that actual error corrected architecture is accomplished at the circuit design level with specific QECC and hardware architecture in mind. Like the classical system, there are several methodologies available to deal with the errors on the circuits.

Recent works have shown that a quantum computer provides exponential speedup than its classical counterpart if the error rate can be kept under certain limit. To deal with errors in the quantum gates, there are many techniques available to implement quantum error correction techniques but such approaches have some design constraints. We have inspired from recent works [14], [15] and these techniques do not have any specific design constraint. We implement such methods to incorporate into our design architecture and provide the result after implementing these algorithms.

Main Contribution: Our work enables a complete design flow for a quantum circuit. First, we prepared a verilog design of the modular exponentiation circuit. From that, we generate optimized quantum gates using synthesis tool RevKit[16]. Our proposed design model for modular exponentiation circuit will pave the way towards large-scale synthesis for quantum circuits. Additionally, for the first time we integrated Quantum Error Correction techniques in the synthesis result. Then we found that different error correction schemes have different performance overheads. We reported it in Table 2 and discussed it in more details in the experimental results and discussion section. Finally, we implemented Linear Nearest Neighbor (LNN) technique proposed in [17] on the top of that.

This paper is ordered as follows. In section II, we describe some basic preliminaries. Section III gives a short introduction on modular exponentiation circuit. In section IV we present the overall design flow of our work. After that in Section V, we have summarized the experimental result. Finally, section VI concludes the work.

II. Basic Preliminaries

A. Boolean Functions

Let $\mathbb{B} = \{0, 1\}$ denotes a *Boolean set* then $\mathcal{B}_{x,y} = \{f : \mathbb{B}^x \rightarrow \mathbb{B}^y\}$ is the set of 2^{y2^x} *Boolean multiple-output function* with *x* inputs and *y* outputs.

Each inputs or literals of Boolean multiple-output function can be either regular or complemented. In logic synthesis, there are several representation of boolean function. *2-level representations*, for example Sum-of-product (SOP) where inputs are combined into product terms using AND operation and product terms are combined using OR. In Exclusive-Sum-Of-Product (ESOP), XOR operation is used instead of OR. *Multi-level representation* are directed acyclic graphs, called logic networks, in which terminal nodes are input variables or constants and internal nodes are logic operations. Several popular instance of homogeneous logic representation includes *AND-Inverter Graphs* (AIG) [18], *Majority Inverter Graphs* (MIG), *XOR Majority Graphs* (XMG). In this work, we use AIG and XMG as logic networks. AIG have AND gates and inverters logic as logic primitives whereas in XMG, XOR, majority operation and inverter are logic primitives.

B. Reversible Circuits

A reversible circuit of size *n* with depth *d* is a cascade of *d* reversible gates over the set of input lines $X = \{x_1, x_2, ..., x_n\}$, i.e., a reversible circuit is denoted as $g_0, g_1, g_2, ..., g_{d-1}$, where each g_i represents the i^{th} reversible gate of the circuit.

The value at target line inverts only when all the values assigned at the control lines are positive and all the values in other line remains unchanged.

C. Quantum Computing

A qubit state $|\psi\rangle$ is modeled as a vector $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ where α and β are complex numbers(or amplitude) which satisfies $|\alpha^2 + |\beta|^2 = 1$. The values $|\alpha|^2$ and $|\beta|^2$ are the probabilities which determines whether the qubit state will be 0 or 1. Here $|0\rangle$ and $|1\rangle$ are the states equivalent to classical bits 0 and 1 but unlike classical bit a qubit is superposition of these two states. A quantum state can also be represented by multiple qubit. For instance, a state involving 2-qubit can be represented by $\alpha|00\rangle + \beta|01\rangle + \gamma|10\rangle + \delta|11\rangle$ which has four amplitude , one for each state $|00\rangle, |01\rangle, |10\rangle, |11\rangle$.

Much akin to classical computing, the quantum operation can also be rendered as circuit. *Quantum circuit* is a description form of a quantum algorithm, and it has the input on the left side, and output on the right. Horizontal lines on the circuit represent the qubit and all operation is performed from left to right with time-ordered set of gates. A simple two-qubit quantum circuit is shown in Fig.1(a) and Fig.1(b) shows a larger quantum circuit. In the given figure the vertical direction signifies space (i.e., number of qubits) and the horizontal direction represents time (i.e., number of quantum operations) and time moves from left to right.

Quantum gates are unitary matrices which performs quantum operations. A matrix U is said to be unitary if it satisfies the operation UU† = U†U = I, where U† specify the conjugate transpose of U (sometimes called Hermitian or adjoint of U), and I is the identity matrix.

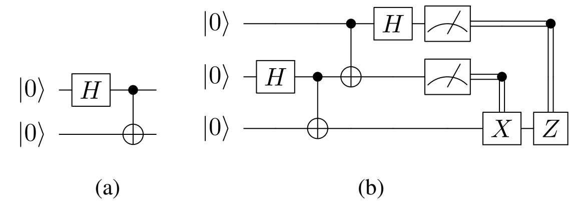

Fig. 1: Example of quantum circuits. Fig.(a)the above gate sequence creates maximally entangles two-qubit state. The output state is$\frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$ Fig.(b) is an example of quantum teleportation circuit

D. Bennett Embedding

Quantum computing needs all operation to be reversible. However, many functions are not reversible in nature so in order to make them reversible we need to perform the embedding process. For this purpose we use *Bennett Trick*[19] which extends an *n* input and *m* output irreversible function into *r* variable reversible function with the help of *r - n* additional lines. *Bennett Trick* is also used to *uncompute* the temporary values from the circuits while performing the computation. For this purpose, we need to double the number of gates and one extra ancillary line for each output.

E. Reversible Logic Synthesis

The process of conversion of classical combinatorial logic into quantum circuits requires reversible logic synthesis. It can be classified into two cases, one *functional algorithm* and another *structural algorithm*. Many functional algorithms are based on *transformation based approach*. Apart from that there is *SAT based approach* and they do not add additional ancillary lines during synthesis, therefore, they give line optimum result.

In *structural algorithm* the input function is in structural representation form, for example, 2-level networks or multilevel logic networks or decision diagrams. The structural approach is more scalable in compare to a functional case but it has a drawback of generating the large set of additional lines.

III. Reversible Modular Exponentiation

Here we have implemented a modular exponentiation circuit in the verilog. In the following, we describe the implementation part in more details.

A modular exponentiation circuit performs the operation *f(x)* = a^x *mod N*, where the value of *a* is lies within 2 and *N-1* and it also co-prime to *N*, whereas the argument of the function *x* takes values between 0 to N^2. The modular exponentiation function can be written as: $a^x modN = (a^{2^0} modN)^{x_0}.(a^{2^1} modN)^{x_1}...(a^{2^{N-1}} modN)^{x_{2N-1}} modN$ where $x_i(i = 0...2N-1)$ are the bits of binary expansion of *x*, that is *x* = $x_{2n-1}, ...x_1, x_0$ and equivalent design is shown in Fig.3.

The technique for computing a^x *(mod)* is stated here below. First, we repeat the operation to measure a^{2^i} *(mod)* for all $i < n$. After that we multiply the powers $x^{2^i}(mod)$ to obtain $a^x(mod)$ where 2^i arise in the binary expansion of *a*. Then we only need to perform $x^a(mod)$ where *a* is evaluated as input in the reversible gate array and *x* is some fixed integer. Here a^i constitute the *i*th bit of *a* in binary representation form, where sequence of bits are ordered in right to left and rightmost bit of *a* is a_0. The method that performs the function is stated here.

```
Data: Input a , x , N
Result: $a^x(modN)$
result = 1;
a = a % N;
while $x > 0$ do
    if $x$ & $1$ then
        result = (result*a) % N;
        x = x >> 1;
        a = a*a % N;
    end
end
```
Algorithm 1: Algorithm for modular exponentiation

Using the above technique it is now straightforward to construct reversible circuit with *i* bit register and *n* bit register that maps the state *(a, y)* into (*a*, $a^x y(modN)$). This can be translated into quantum circuit using O(n^3) gates and the corresponding quantum transformation is $|a\rangle\, |y\rangle \rightarrow |a\rangle\, a^x y(modN)$.

After preparation of the reversible modular exponentiation function in verilog, the output is fed to the next level and the complete description is described below in the next section.

Example 1: Let *x* = 2, *a* = 3, *N* = 5 . We have 2^3 *y*(mod 5) = 3. In verilog, we get the output $(00000011)_2$ with 8 bit width. Hence, the output *y* = $2^3 y(mod5) = 3$.

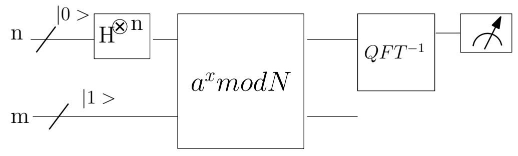

Fig. 2: An outline of the Shor's algorithm, the square block on the left performs modular exponentiation is set of classical gates, other hand quantum part including QFT circuits and hadamard gates

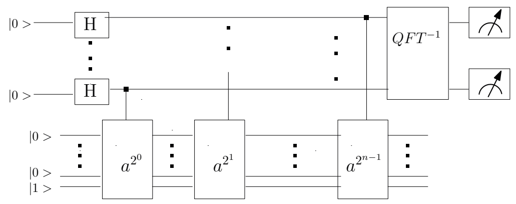

Fig. 3: Design of modular exponentiation circuit performing modular multiplication over each multiplier block. Each multiplier block performs a modular multiplication on the lower register.

IV. Design Flow

In this section, we discuss the overall design flow of our work. Fig.4 illustrates the design flow consists of several constituent layers. First, at the design level, we have a verilog description of the exponentiation function. In the logic synthesis level the verilog design is transformed and optimized into an intermediated representation form *Xor Majority Graph* (XMG). After that in Reversible synthesis level, these intermediate representation form generates reversible circuits using the synthesis tool RevKit[16] which further passed to next level, quantum level. At this level, we perform different error correction methods on the circuits so the resulting circuits become error protected circuit then finally to make the circuits more efficient we apply LNN optimization process. All the details are described hereafter.

A. Direct XMG based synthesis

The input to this level is *Xor Majority Graph* (XMG) that is generated after parsing the verilog file through synthesis tool ABC. First, we generate the optimized *And Inverter Graph* (AIG) through ABC tool using some commands then

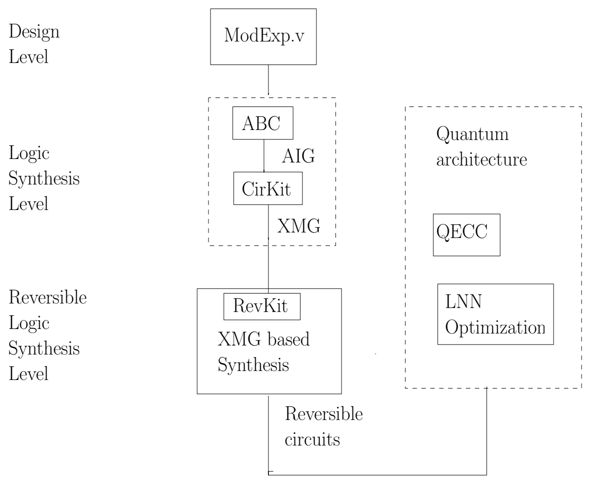

Fig. 4: Illustration of work-flow

we read the AIG file into RevKit and convert it into *Xor Majority Graph*. An XMG[20] is a data structure in which basic operations are AND, OR, XOR and MAJ (the majority of three functions) and their complemented forms. This networks representation has a certain advantage when we perform synthesis operation of boolean function into quantum circuits. Firstly, the MAJ operation can be performed with only one Toffoli gate which means it has the same number of T gates as an AND and OR gate and an XOR gate can be realized using single CNOT gate so it does not require any T gates. Apart from that the XOR gate can be applied in-place operation which means if the value it holds is no longer necessary to compute another gate. The same logic applies for the MAJ gate also if all of its operands values are no longer necessary. XMG files are optimized using the CirKit command '*xmglut -k 4*'. Then we perform XMG based synthesis to generate reversible circuits using the CirKit command '*dxs*'.

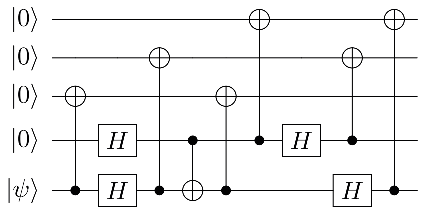

Fig. 5: Optimal encoding circuits for the five-qubit code.

B. QECC

In this phase, we incorporate Quantum Error Correcting Codes (QECC) on the optimized circuits. Unlike classical computer in the quantum system, different error correction mechanism is required to protect the quantum superposition and entanglement. Here we use two different techniques and compare the effect of applying these methods. In first case, we apply 5-qubit encoding circuit [14] as in 5 to protect the qubits from errors. This five-qubit ([5, 1, 3] code) code corrects all 1-bit errors with the minimum number of additional qubits. So, for the five-qubit code we took optimal encoding circuits which encode the original state by disseminating quantum information over five qubits. The encoder circuit takes the initial state with four additional qubits in the state $|0\rangle$ to the encoded state. This encoder circuit can be transformed into a decoding circuit if we run it backward. Reading the extra four qubits at the decoder's output we can recognize, out of the sixteen possible alternatives (no error and all fifteen possible 1-bit errors), which error has been generated. More details in the QECC technique can be found in the literature as we mentioned above.

In the second technique, a different approach is employed on the circuit in which we detect the types of errors and the places, where the errors occur. In Fig. 6 q_0 to q_4 represent data-bits and q_5 to q_8 represent ancillary qubits and the ancillary qubits q_5, q_6 check for bit errors whereas q_7, q_8 check for the phase errors. The error is detected when the first four and the last four qubits are not in the same state. So, instead of the error correcting block, error detection block is placed after a certain time interval. If we detect any error then the correction block as shown in Fig. 7 is placed at the location.

C. Linear Nearest Neighbor (LNN) Optimization

After implementing the QECC parts, the final version of the circuits is optimized with LNN architecture using the CirKit[1]tool. The reason is that in the optimized quantum circuits in CNOT gates control bits and target bits are not adjacent so thereby increasing NNC (Nearest Neighbor Cost) cost[2]and if there are large numbers of CNOT gates exist in the circuit then overall cost parameter will be relatively high. To make the circuits further cost efficient they must satisfy the nearest neighbor constraint property. In general, by incorporating some more gates we can make a quantum circuit in LNN optimized forms. In our works, we have incorporated the LNN property by keeping in mind the usefulness and implementing it in quantum hardware.

TABLE I: Comparison results of modular exponentiation

	XMG based synthesis		Result from [11]		%improvement	
i/p	Max T-cost	Avg. T-cost	Max T-cost	Avg. T-cost	Max T-cost	Avg. T-cost
7	182	134.292	192	147.46	-5%	-9%
8	257	194.304	272	197.28	-5%	-1%
9	330	258.041	334	268.67	-3%	-0.3%
10	421	327.479	418	318.34	1%	2%
11	526	405.386	515	397.25	2%	2%
12	645	489.286	620	470.56	4%	4%
13	764	580.808	741	560.62	3%	3%
14	910	610.342	881	578.13	3%	5%

[1]github.com/msoeken/cirkit

[2]A quantum gate Q(c,t) where c is control bit and t is target bit then $|c - t| - 1$ is called NNC cost of the quantum gate.

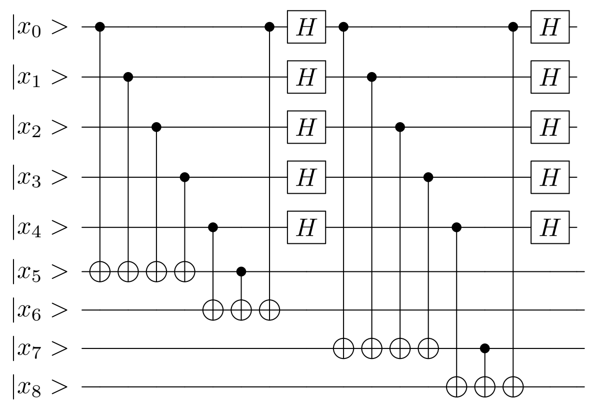

Fig. 6: 5-qubit Error detection circuit

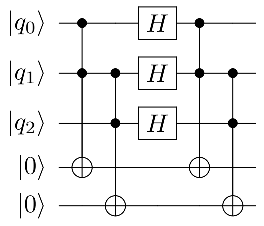

Fig. 7: 3-qubit Error correction circuit.

V. Experimental Results and Discussion

We have generated the optimized quantum circuit for the exponentiation using the above-mentioned design flow in section IV. We used ABC[18] and Revkit[16] to generate the intermediate representation, XMG which is used to perform synthesis to generate reversible quantum circuits. Furthermore, Quantum Error Correcting Codes (QECC) and Linear Nearest Neighbor (LNN) property have been incorporated in the resulting circuits. All the operation have been carried out on an Intel Core i5-2500 4 core CPU with 3.30 GHz and 8 GB main memory.

In Table 1 we have compared our result with [11]. Here we have considered the result after performing XMG based synthesis without considering error correction technique to compare with that result[3]. We have shown two parameters Max(Maximum) T-cost and Avg.(Average) T-cost. For a particular bit width of *x* of the function *f(x)* = $a^x modN$ we took different values of a and N. For example, when x = 7 and N = 65 we have found T-cost = 192 of the circuit according to our result. Again for same bit width but different values of a and N we have calculated T-cost and from that results we have shown Max T-cost and Avg. T-cost.

Table 2 lists the experimental result for modular exponentiation circuit. We reported the number of qubits or no. of ancillary lines, T_c (number of T gates or T-count) and number of gates etc for each circuit. The first column in Table 2 gives benchmark specification which is bit width of *x* of the function *f(x)* = $a^x modN$ and next three columns (no. of lines, gate-count, T-count respectively) highlight the results after performing Direct XMG based synthesis. Like Table 1 we have calculated the T-count value where the values of a and N are chosen randomly.

In the next phase, we employ two error correcting methods on the result-set that we found in previous columns in Table 2. The next six columns show the result after applying error correction techniques in two different methods. If we compare this two methods then we can see that cost parameter (number of ancillary lines, T-count) in first method is relatively higher than the second method. This is due to the fact that resource requirements for the computation have been minimized in the second strategy. In this technique after locating the errors, the error correction blocks have been placed. The final circuits are optimized with LNN property in both cases.

In comparison with the normal synthesis approach, introducing Quantum Error Correction technique leads to higher resource requirements. It can be seen from the table 1 that after applying error-correcting parts the number of qubits (or ancillary lines) increase significantly. This occurs because while we add encoding part with the data bits to protect single bit data 5 additional lines are introduced so thereby increasing the number of lines. After that introducing LNN parts also increases the number of lines in the circuits but it minimizes NNC cost in the circuits. In particular, in terms of circuit efficiency, it is necessary to integrate QECC parts in the quantum circuits and LNN parts make the circuits more design efficient for quantum architecture.

There is always a trade-off between the number of ancillary lines and T-count and in our case, we performed T-count optimized design. We can also reduce the number of lines in the circuit using line optimization method in RevKit but compromise T-count. As discussed above, for a function such as exponentiation function has a very important role in Shor's algorithm, so this design is significant in this regard.

VI. Conclusions

We reported a unique design flow for the synthesis of modular exponentiation functions in the quantum computer. Our circuit can be used to obtain resource estimates for various quantum algorithms and the results may help to identify the first large-scale applications, as well as bottlenecks in these algorithms where more research is necessary in order to make the resource requirements practical. When integrated with quantum compilation tool our exponentiation functions will be very useful for automatic code generation while performing oracle that calculates this function. This tremendously facilitates the implementation of quantum algorithms which employ oracles that compute such functions on a superposition of inputs.

While employing QECC codes in the quantum circuits our methods allow to reduce the Toffoli and qubit counts significantly, the resulting circuits are still quite expensive,

[3]web.eecs.umich.edu/ imarkov/MME/

TABLE II: Results for modular exponentiation circuit

Benchmark specification	Direct XMG based synthesis			Applying 5-qubit code from [14]			applying QECC using [15]			%improvement in [15] over [14]		
i/ps	no. of qubits	gates	T_c	no. of lines	gates	T_c	no. of qubits	gates	T_c	no. of lines	gates	T_c
7	76	132	192	96	220	582	88	210	526	9	4	10
8	84	146	280	114	354	699	102	334	661	11	5	5
9	98	166	379	128	188	982	112	178	892	7	5	11
10	102	174	426	134	196	1098	121	187	1056	10	7	3
11	102	174	576	145	220	1296	131	204	1256	10	7	3
12	112	187	634	156	234	1378	144	220	1342	8	6	2
13	126	199	765	198	201	1578	172	195	1467	15	3	7
14	156	235	834	265	802	1925	220	605	1426	20	32	35
15	201	320	1125	341	1123	2428	301	991	1876	13	13	29
16	273	488	1591	464	1323	2928	401	1191	2176	15	11	23
17	476	601	2303	809	1676	3978	741	1365	3126	9	22	21
18	576	810	4312	680	1824	4778	620	1667	4278	9	22	21
19	712	1031	5165	1210	2676	6174	1078	2365	5418	12	13	16
20	982	1967	9123	1669	3930	11447	1554	2997	9836	7	21	17
21	1137	2879	12181	1932	7167	17494	1667	5887	14832	15	21	22
22	1446	3534	17854	2458	9134	23178	2231	7991	19988	10	14	15
23	1667	5135	23453	2833	15184	31876	2674	11991	28964	5	26	10
24	1952	8635	34323	3318	21032	44078	3210	17674	39624	3	19	8
25	2271	14635	69343	3860	34752	88278	3650	30647	80654	5	13	9
26	2561	20645	76123	4353	44660	92621	4125	41564	81524	5	7	13
27	2765	32314	91192	4719	62680	99454	4167	57345	90548	13	9	9
28	2961	49314	98192	5033	72680	112454	4911	69345	100545	3	4	11
29	3270	59512	102533	5559	67561	122581	5147	64463	111643	8	8	9
30	3480	71368	114422	5916	92328	138220	5641	87234	128212	4	10	8
31	3974	78118	123533	6755	101782	156861	6448	166775	141652	5	7	11
32	4281	81368	134732	7277	128328	182220	7112	112234	167834	3	6	8
Average %improvement in [15] over [14]										**8%**	**10%**	**11%**

especially in terms of the number of gates and ancillary lines that are required. As the inclusion of quantum error-correcting codes in the quantum circuits while performing synthesis is new and challenging area and some recent works have been done in this regard. We hope that more research in the implementation of quantum error-correcting codes is necessary in order to further reduce the cost originating from the circuit in the computational basis.

REFERENCES

[1] P. W. Shor, "Algorithms for quantum computation: Discrete logarithms and factoring," in *Foundations of Computer Science, 1994 Proceedings., 35th Annual Symposium on*. Ieee, 1994, pp. 124–134.

[2] E. A. Martinez, C. A. Muschik, P. Schindler, D. Nigg, A. Erhard, M. Heyl, P. Hauke, M. Dalmonte, T. Monz, P. Zoller *et al.*, "Real-time dynamics of lattice gauge theories with a few-qubit quantum computer," *Nature*, vol. 534, no. 7608, pp. 516–519, 2016.

[3] P. O'Malley, R. Babbush, I. Kivlichan, J. Romero, J. McClean, R. Barends, J. Kelly, P. Roushan, A. Tranter, N. Ding *et al.*, "Scalable quantum simulation of molecular energies," *Physical Review X*, vol. 6, no. 3, p. 031007, 2016.

[4] N. Yoran and A. J. Short, "Classical simulability and the significance of modular exponentiation in shor's algorithm," *Physical Review A*, vol. 76, no. 6, p. 060302, 2007.

[5] T. G. Draper, "Addition on a quantum computer," *arXiv preprint quant-ph/0008033*, 2000.

[6] S. Dutta, D. Bhattacharjee, and A. Chattopadhyay, "Quantum circuits for toom-cook multiplication," *arXiv preprint arXiv:1805.02342*, 2018.

[7] M. K. Bhaskar, S. Hadfield, A. Papageorgiou, and I. Petras, "Quantum algorithms and circuits for scientific computing," *arXiv preprint arXiv:1511.08253*, 2015.

[8] T. Häner, M. Roetteler, and K. M. Svore, "Optimizing quantum circuits for arithmetic," *arXiv preprint arXiv:1805.12445*, 2018.

[9] T. Häner, M. Soeken, M. Roetteler, and K. M. Svore, "Quantum circuits for floating-point arithmetic," *arXiv preprint arXiv:1807.02023*, 2018.

[10] A. Pavlidis and D. Gizopoulos, "Fast quantum modular exponentiation architecture for shor's factorization algorithm," *arXiv preprint arXiv:1207.0511*, 2012.

[11] I. L. Markov and M. Saeedi, "Constant-optimized quantum circuits for modular multiplication and exponentiation," *arXiv preprint arXiv:1202.6614*, 2012.

[12] M. Amy, D. Maslov, and M. Mosca, "Polynomial-time t-depth optimization of clifford+ t circuits via matroid partitioning," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 33, no. 10, pp. 1476–1489, 2014.

[13] M. Soeken, M. Roetteler, N. Wiebe, and G. De Micheli, "Design automation and design space exploration for quantum computers," in *Proceedings of the Conference on Design, Automation & Test in Europe*. European Design and Automation Association, 2017, pp. 470–475.

[14] V. Kliuchnikov and D. Maslov, "Optimization of clifford circuits," *Physical Review A*, vol. 88, no. 5, p. 052307, 2013.

[15] R. Majumdar, S. Basu, and S. Sur-Kolay, "A method to reduce resources for quantum error correction," in *International Conference on Reversible Computation*. Springer, 2017, pp. 163–175.

[16] M. Soeken, S. Frehse, R. Wille, and R. Drechsler, "Revkit: A toolkit for reversible circuit design." *Multiple-Valued Logic and Soft Computing*, vol. 18, no. 1, pp. 55–65, 2012.

[17] M. Saeedi, R. Wille, and R. Drechsler, "Synthesis of quantum circuits for linear nearest neighbor architectures," *Quantum Information Processing*, vol. 10, no. 3, pp. 355–377, 2011.

[18] R. Brayton and A. Mishchenko, "Abc: An academic industrial-strength verification tool," in *International Conference on Computer Aided Verification*. Springer, 2010, pp. 24–40.

[19] C. H. Bennett, "Logical reversibility of computation," *IBM journal of Research and Development*, vol. 17, no. 6, pp. 525–532, 1973.

[20] W. Haaswijk, M. Soeken, L. Amaru, P.-E. Gaillardon, and G. De Micheli, "A novel basis for logic rewriting," in *Design Automation Conference (ASP-DAC), 2017 22nd Asia and South Pacific*. Ieee, 2017, pp. 151–156.