Reversible Logic Synthesis of k-Input, m-Output Lookup Tables

Alireza Shafaei, Mehdi Saeedi, Massoud Pedram
Department of Electrical Engineering
University of Southern California, Los Angeles, CA 90089
{shafaeib, msaeedi, pedram}@usc.edu

***Abstract*—Improving circuit realization of known quantum algorithms by CAD techniques has benefits for quantum experimentalists. In this paper, we address the problem of synthesizing a given k-input, m-output lookup table (LUT) by a reversible circuit. This problem has interesting applications in the Shor's number-factoring algorithm and in quantum walk on sparse graphs. For LUT synthesis, our approach targets the number of control lines in multiple-control Toffoli gates to reduce synthesis cost. To achieve this, we propose a multi-level optimization technique for reversible circuits to benefit from shared cofactors. To reuse output qubits and/or zero-initialized ancillae, we uncompute intermediate cofactors. Our simulations reveal that the proposed LUT synthesis has a significant impact on reducing the size of modular exponentiation circuits for Shor's quantum factoring algorithm, oracle circuits in quantum walk on sparse graphs, and the well-known MCNC benchmarks.**

***Keywords*-Lookup tables; Logic synthesis; Reversible circuits; Shor's quantum number-factoring algorithm; Binary welded tree.**

I. Introduction

Quantum information processing has captivated atomic and optical physicists as well as theoretical computer scientists by promising a model of computation that can improve the complexity class of several challenging problems [1]. A key example is Shor's quantum number-factoring algorithm which factors a semiprime M with complexity $O((\log M)^3)$ on a quantum computer. The best-known classical factoring algorithm, the *general number field sieve*, needs $O(e^{(\log M)^{1/3}(\log\log M)^{2/3}})$ time complexity. Other quantum algorithms with superpolynomial speedup on a quantum computer include quantum algorithms for discrete-log, Pell's equation, and walk on a binary welded tree [2].

Improving circuit realization of known quantum algorithms — the focus of this work — is of a particular interest for lab experiments. In 2000, researchers implemented Shor's number-factoring algorithm to factor the number 15 [3]. In March 2012, physicists published the first quantum algorithm that can factor a three-digit integer, 143 [4]. CAD algorithms and tools are required to help with physical circuit realization even for a few number of qubits and gates. For example, a previous method in [5] required at least 14 qubits to factor the number 143. This exceeds the limitation of current quantum computation technology. Accordingly, [4] introduced an optimization approach to reduce the number of total qubits.

978-3-9815370-0-0/DATE13/©2013 EDAA

In this paper, we propose an automatic technique to synthesize a specific type of quantum circuits that has applications in, at least, quantum circuits for number factoring and quantum walk [6]. In particular, we aim to synthesize a given *lookup table* (LUT) by reversible gates. Following [7], a (k, m)-lookup table takes k read-only inputs and $m > \log_2 k$ zero-initialized ancillae (outputs). For each 2^k input combination, a (k, m)-LUT produces a pre-determined m-bit value. In [7], Markov and Saeedi showed LUT synthesis can improve modular exponentiation circuits for Shor's algorithm. In this paper, we will show LUT synthesis can also improve practical implementation of a quantum walk on graphs. Additionally, we will discuss how LUT synthesis can improve the costs of irreversible benchmarks. The rest of the paper is organized as follows. In Section II, basic concepts are introduced. Section III presents different applications for LUT synthesis. Related works are discussed in Section IV. We propose LUT synthesis approach in Section V. Experimental results are given in Section VI, and finally Section VII concludes the paper.

II. Basic concept

Boolean Logic. The set of n variables of a Boolean function is denoted as $x_0, x_1, \cdots, x_{n-1}$. For a variable x, x and $\bar{x}$ are *literals*. A Boolean product, *cube*, is a conjunction of literals where x and $\bar{x}$ do not appear at the same time. A *minterm* is a cube in which each of the n variables appear once, in either its complemented or un-complemented form. A sum term in which each of the n variables appears once is called a *maxterm*. A *sum-of-product* (SOP) Boolean expression is a disjunction (OR) of a set of cubes. A *product-of-sum* (POS) expression is a conjunction (AND) of maxterms. An *exclusive-or-sum-of-product* (ESOP) representation is an XOR (modulo-2 addition) of a set of cubes. For a given function, the subfunction which results from replacing a variable by 1 (for sum-of-product) or 0 (for product-of-sum) is called a *cofactor*. For a finite set A, a one-to-one and onto (bijective) function $f : A \to A$ is a *permutation*, which is called a *reversible function*. Among $\sum_{i=1}^{n} (2^i)^{2^n} \simeq 2^{n2^n}$ irreversible multiple-output (from 1 to n) functions, $2^n!$ distinct reversible functions exist. To convert an irreversible specification to a reversible function, input/output should be added.

Quantum Bit and Register. A quantum bit, *qubit*, can be treated as a mathematical object that represents a quantum state with two basic states $|0\rangle$ and $|1\rangle$. It can also carry a linear combination $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ of its basic states,

called a *superposition*, where α and β are complex numbers and $|\alpha|^2$+$|\beta|^2$=1. Although a qubit can carry any norm-preserving linear combination of its basic states, when a qubit is *measured*, its state collapses into either $|0\rangle$ or $|1\rangle$ with probabilities $|\alpha|^2$ and $|\beta|^2$, respectively. A *quantum register* of size n is an ordered collection of n qubits. Apart from the measurements that are commonly delayed until the end of a computation, all quantum computations are reversible.

Quantum Gates and Circuits. A matrix U is *unitary* if $UU^\dagger = I$ where $U^\dagger$ is the conjugate transpose of U and I is the identity matrix. An n-qubit *quantum gate* is a device which performs a $2^n \times 2^n$ unitary operation U on n qubits in a specific period of time. For a gate g with a unitary matrix U_g, its inverse gate g^{-1} implements the unitary matrix U_g^{-1}. A reversible gate/operation is a 0-1 unitary, and reversible circuits are those composed with reversible gates. A reversible gate realizes a reversible function. A *multiple-control Toffoli gate* C^{n}NOT $(x_1, x_2, \cdots, x_{n+1})$ passes the first n qubits unchanged. These qubits are referred to as *controls*. This gate flips the value of $(n+1)$st qubit if and only if the control lines are all one (positive controls). Therefore, action of the multiple-control Toffoli gate may be defined as follows: $x_{i(out)} = x_i (i < n+1), x_{n+1(out)} = x_1 x_2 \cdots x_n \oplus x_{n+1}$. Negative controls may be applied similarly. For $n = 0$, $n = 1$, and $n = 2$ the gates are called NOT, CNOT, and Toffoli, respectively. The lines which are added to make an irreversible specification, reversible are named *ancillae* which normally start with 0. The zero-initialized ancillae may be modified inside a given subcircuit, but should be returned to zero at the end of computation to be reused.

Cost Model. Quantum cost (QC) is the number of NOT, CNOT, and controlled square-root-of-NOT gates required for implementing a given reversible function. QC of a circuit is calculated by a summation over the QCs of its gates. In addition to the QC model, a single-number cost based on the number of two-qubit operations required to simulate a given gate was proposed in [8]. This model captures the complexity of physical implementation of a given gate based on the Hamiltonian describing the underlying quantum physical system. In particular, it estimates the cost of a C^{n}NOT (and $n \geq 2$) as $2n - 5$ 3-qubit Toffoli gates (and $10n - 15$ 2-qubit gates).

III. Possible applications of LUT Synthesis

Specific reversible circuits must be motivated by applications [9]. In the following, we introduce several applications of LUT synthesis in quantum computation.

A. Quantum Algorithm for Number Factoring

Shor's quantum number factoring uses quantum circuit for modular exponentiation $b^x \% M$ (% is modulo operation) for a semiprime $M = pq$ for primes p and q and a randomly selected number b. Modular exponentiation is performed by n conditional modular multiplications $Cx\%M$ where C and M are coprime. Precisely, for the binary expansion $x = x_n 2^n + x_{n-1} 2^{n-1} + \ldots + x_0$ (and x_i is 0 or 1), $b^x \% M = b^{x_n 2^n} \times b^{x_{n-1} 2^{n-1}} \times \ldots \times b^{x_0} \% M$. Hence, one needs to implement multiplication by $b^{2^n} \% M$ conditioned on x_n, multiplication by $b^{2^{n-1}} \% M$ conditioned on $x_{n-1}, \ldots$, and multiplication by $b\%M$ conditioned on x_0, in sequence. In [7, Section 7.2], the authors introduced one (k, m)-LUT (for $k = 4$) to implement the (four) most expensive conditional modular multiplications that appear in modular exponentiation to reduce total cost. For example [7, Figure 15], implemented conditional modular multiplications by 4, 16, 82, and 25 in modular exponentiation for $b = 2$, $M = 87 = 3 \times 29$ by a systematic method. The related outputs of this (4,7)-LUT are 1, 4, 16, 64, 82, 67, 7, 28, 25, 13, 52, 34, 49, 22, 1, and 4 which results from considering different combinations (by multiplication) of 4, 16, 82, and 25 %87. Except for the four most expensive modular multiplications, other modular multiplications are implemented directly in [7]. In this work, we propose an automatic LUT synthesis method that can further improve modular exponentiation circuits.

B. Quantum Walk for Sparse Graphs

In [10, Thereom 1], the authors proposed a polynomial-size circuit for quantum walk on a sparse graph with 2^n nodes and with adjacency matrix P. A graph is *sparse* if each node has at most d transitions (or edges) to other nodes. To propose the circuit, the authors assumed **(1)** there is a polynomial-size reversible circuit returning the list of (at most d) n-bit neighbors of the node x according to P **(2)** there is a polynomial-size reversible circuit returning the list of (at most d) t-bit precision transition probabilities. Our LUT synthesis can be used to construct circuits for (1) and (2).

C. Quantum Walk on Binary Welded Tree

As a special case of quantum walk on sparse graphs, one can consider a binary welded tree. A *binary welded tree* (BWT) is a graph which consists of two binary trees that are *welded* together with a random function between the leaves. Fig 1-a shows a sample BWT. In a BWT every node has degree three except the root of each tree (which has degree two). A BWT has $2(2^{n+1} - 1)$ nodes for a binary tree of height n. Therefore, strings of $m > \lceil \log_2 2(2^{n+1} - 1) \rceil$ bits are required to represent each node uniquely (minimum m is $n + 2$). All edges of a node in a BWT are uniquely colored and each color is denoted by c. The number of colors used in a BWT is at least 3 and at most 4 (by Vizing's theorem for graph coloring).

In [6], Childs et al. proposed an *oracle-based* quantum walk algorithm on BWT that is exponentially faster, with $O(n)$ oracle queries, on a quantum computer than on a classical computer. The best-known classical algorithm needs O(2^n) oracle queries. The oracle function $v_c(a)$ takes as input the node label a and an edge color c, and returns the label $v_c(a)$ of a node that is connected to node a. As an example for the BWT in Fig. 1-a and c=black, we have (Fig. 1-b) $v_c(7) = 16, v_c(8) = 17, v_c(9) = 15, v_c(11) = 19, v_c(12) = 22, v_c(13) = 18, v_c(14) = 20$ (and vice versa, e.g., $v_c(16) = 7$).[1] If there is no connection to a with color c, the oracle returns the unique label `invalid`. In [6], this unique value is all

[1]Permutations in BWT include 2-cycles. For a synthesis algorithm that extensively works with cycles see [11].

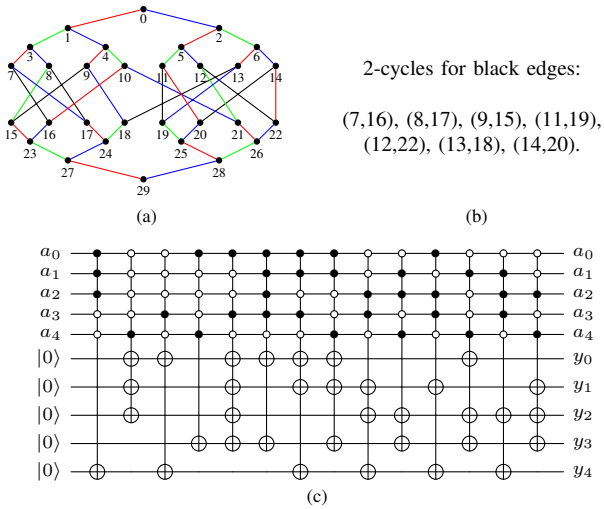

Fig. 1: (a) A sample binary welded tree. (b) Lookup table of the oracle for black edges. A 2-cycle (a, b) is a permutation which exchanges two elements and keeps all others fixed. (c) An oracle implementation. In general, one needs l C^{k}NOT gates to implement each minterm where l is the number of bits with value 1 in the binary representation of the minterm. For example, the first gate implements 16 (i.e., "10000" in binary) for 7 (i.e., "00111" in control lines — two negative and three positive controls). The second gate implements 7 (i.e., "00111" which needs three target lines) for 16 (i.e., "10000" in control lines). Other gates can be constructed similarly.

ones. Outputs should be constructed on a septate register so that input register remains unchanged for future queries. Note that in a physical implementation, besides the number of queries to the oracle, the computation performed by the oracle also affects runtime. Accordingly, we use LUT synthesis to improve the physical implementation of a given oracle circuit.

IV. Related work

A trivial approach for LUT synthesis is to implement each input combination of a (k, m)-LUT with at most m C^{k}NOT gates. For example, reconsider the BWT in Fig. 1-a where the circuit in Fig. 1-c constructs the oracle. To handle the `INVALID` label, initialize outputs to all ones and flip target locations in Fig. 1-c. However, large number of Toffoli gates with many controls are expensive for physical implementation.

ESOP-based approaches [12], [13] are fast and are able to handle large sizes of both reversible and irreversible functions. The basic idea is to write each output as an ESOP representation and implement each term by a multiple-control Toffoli gate [12]. In the recent years, several improved ESOP-based approaches, e.g., [13], have been proposed which use shared product terms (cubes) to reduce the number of Toffoli gates. However, these approaches usually lead to expensive multiple-control Toffoli gates with many controls.

Reversible logic synthesis methods [9] can also be used to synthesize a given (m, m)-LUT. To this end, input register should be copied (by m CNOT gates) into output register so that inputs remain unchanged. However, these approaches are general and may not exploit LUT structures for cost reduction. Other approaches are based on Davio decompositions[2] which include the method in [7] for $(4, m)$-LUT synthesis and the method in [14]. Method in [7] uses cofactors for multi-level optimization in logic synthesis but it is limited to $(4, m)$-LUT implementation. By assuming that the factors have already been computed on dedicated ancillae, [14] implements the Davio decompositions. It leads to numerous ancillae.

V. The Proposed Synthesis Algorithm

Multi-level logic synthesis for irreversible functions has a rich history. However, conventional logic-synthesis approaches cannot be immediately used for cofactor extraction and multi-level circuit realization in reversible circuits. Basically, in a multi-level implementation of a set of functions, it is allowed to use an *unlimited* number of intermediate signals. This is due to the fact that intermediate signals in classical circuits can be realized with low cost. However, in quantum circuits each intermediate signal should be constructed on one qubit[3] and the number of qubits in current quantum technologies is very limited. In this section, we proposed some techniques to reduce the number of ancillae required in a multi-level optimization.

Un-computation. A common approach to exploit cofactors in circuit realization for reversible circuits is to construct an intermediate signal on a zero-initialized ancilla and use it to optimize different outputs. This process should be followed by *un-computing* the constructed cofactor to recover the zero-initialized ancilla for future use. The reason for un-computation is twofold. **(1)** Without un-computation, each cofactor needs a new ancilla (qubit) and the number of available qubits is very restricted in current quantum technologies. **(2)** Constructing a zero state from an unknown quantum state *generally* needs an exponential number of gates [15].

Cube sharing. As done in [13], common cubes among different functions may be shared to avoid multiple constructions of the same cube. It can be performed by constructing the shared cube once and copying the result by several CNOTs. For a reversible function with several outputs, each cube appears at least once in one output. So, it is possible to construct this cube on the related output line. Cube sharing can reduce the number of Toffoli gates, but it leaves the number of controls as is. The recent ESOP-based optimization methods for reversible circuits, e.g., [13], restrict circuit optimization to use only the cubes which exist in ESOP representation of a given function. However, their performance can be limited. For example, consider $y_0 = ab$ and $y_1 = abc$. Note that each cube appears once. Therefore, no cube can be shared. Fig. 2-a shows a circuit with one C^{2}NOT and one C^{3}NOT.

Cofactorization. Relaxing the constraint of sharing available cubes promises a significant cost reduction. As an example for the circuit shown in Fig. 2-a, it is possible to reuse the cofactor ab twice. This can be done by constructing the cofactor ab on y_0 (Fig. 2-b), and reusing it to construct abc on y_1. For a given function, the number of possible cofactors

[2]Positive Davio and negative Davio decompositions are defined by $f = f_{x_i=0} \oplus x_i . f_{x_i=2}$ and $f = f_{x_i=1} \oplus \bar{x}_i . f_{x_i=2}$ for $f_{x_i=2} = f_{x_i=0} \oplus f_{x_i=1}$.

[3]Recall that reversible functions are unitary transformation. As a result, explicit fanouts and loops/feedback are prohibited.

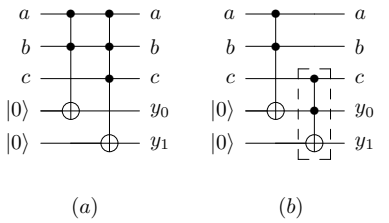

Fig. 2: Circuits for $y_0 = ab, y_1 = abc$, (a) without cofactor sharing, (b) with cofactor sharing.

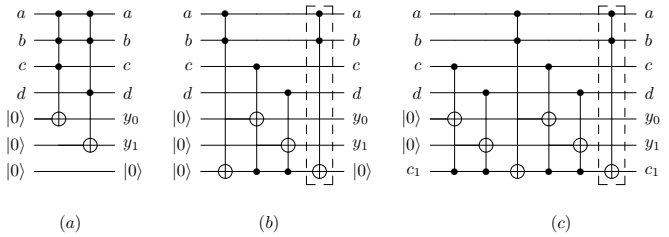

Fig. 3: Circuits for function $y_0 = abc, y_1 = abd$. (a) Initial circuit. (b) An equivalent circuit constructed by reusing ab as a shared cofactor. (c) Circuit in (a) when no zero-initialized *extra* qubit exists. Gates in dashed box are used to un-compute the cofactor ab.

can be very large. Accordingly, finding the most appropriate set of cofactors is a challenging problem.[4]

Copying. A shared cofactor can be constructed on a zero-initialized ancilla by a subcircuit C. To reuse the ancilla, one needs to un-compute the constructed cofactor by applying C^{-1} (the inverse[5] of C). As an example, consider $y_0 = abc$, $y_1 = abd$. Fig. 3-a shows the circuit. As done in Fig. 3-b, one can temporarily construct the cofactor ab on a zero-initialized ancilla (the first gate), and use it to construct dependent cubes (gates #2 & #3). The constructed cofactor is un-computed finally. Generally, following this path leads to an optimized circuit but it adds an ancilla. To overcome, we use output lines with any arbitrary Boolean value to construct cofactors by adding one extra gate. Consider Fig. 4-a with two qubits with initial values c_1 and $|0\rangle$. Assume that f and g are two cofactors (their actual circuits are not shown) and the goal is to construct $c_1 \oplus fg$ on the first qubit. Fig. 4-a illustrates the circuit by constructing the cofactor f. Now, assume that the value in the second qubit is any arbitrary Boolean value c_2. To remove the effect of c_2, we add one extra gate before constructing the cofactor f. See Fig. 4-b for detail. Un-computation can be done by reapplying the circuit for f. Fig. 3-c shows an example for the circuits in Fig. 3-a and Fig. 3-b. Clearly, for $g = 1$ (which leads to applying CNOT for gates which use g), circuits in Fig. 4 copy f from the second qubit to the first qubit. For other nontrivial cases, this circuit may be a *generalized* copying circuit.

Cofactor list. For a (k, m)-LUT with input variables x_i and output variables y_j ($0 \le i < k$, $0 \le j < m$), we use a row vector $[\alpha_0, \ldots, \alpha_{k-1}, \beta_0, \ldots, \beta_{m-1}]$ to represent a cube

[4]Constructing cofactors may need un-computation. This problem is more challenging in reversible logic as compared to its conventional counterpart.

[5]To implement C^{-1} for C with only multiple-control Toffoli gates, one needs to apply gates in the reverse order.

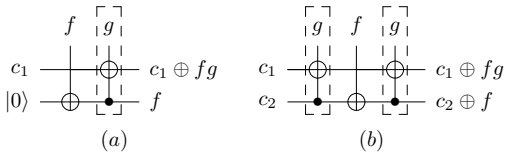

Fig. 4: Copying a cofactor by at most two gates, (a) with a zero-initialized ancilla, (b) without a zero-initialized ancilla.

TABLE I: `cube_list` for the (3,6)-LUT given in Example 5.1.

Cube $\mathcal{C}$	α_0	α_1	α_2	β_0	β_1	β_2	β_3	β_4	β_5
$x_0'x_1'x_2'$	0	0	0	1	0	0	0	0	0
$x_0'x_1x_2$	0	1	1	1	1	1	1	0	0
$x_0x_1'x_2$	1	0	1	1	0	0	0	0	0
$x_0x_1'x_2'$	1	0	0	0	1	0	1	0	0
$x_0'x_1x_2'$	0	1	0	0	0	1	0	0	1
$x_0x_1x_2$	1	1	1	0	0	1	0	0	1
$x_0'x_2$	0	2	1	0	0	0	0	1	0
x_0x_2'	1	2	0	0	0	0	0	1	0

TABLE II: `shared_cofactor_list` for Example 5.1.

Shared cofactor	Frequency	Dependent cubes
$x_1'x_2'$	3	$x_0'x_1'x_2', x_0x_1'x_2'$
$x_0'x_2'$	3	$x_0'x_1'x_2', x_0'x_1x_2'$
$x_0'x_1$	6	$x_0'x_1x_2, x_0'x_1x_2'$
x_1x_2	6	$x_0'x_1'x_2', x_0x_1'x_2'$
$x_0'x_2$	5	$x_0'x_1x_2, x_0'x_2$
x_0x_1'	3	$x_0x_1'x_2, x_0x_1'x_2'$
x_0x_2	3	$x_0x_1'x_2, x_0x_1x_2$
x_0x_2'	3	$x_0x_1'x_2', x_0x_2'$

$\mathcal{C}$ [16, Section 2.3]. In this notation, $\alpha_i = 0$ if x_i appears as complemented, $\alpha_i = 1$ if x_i appears as un-complemented, and $\alpha_i = 2$ if x_i does not exist in $\mathcal{C}$. Additionally, $\beta_j = 0$ if $\mathcal{C}$ is not available in y_j, and $\beta_j = 1$ if $\mathcal{C}$ is available in y_j. We use a tabular format, `cube_list`, to store all cubes. For n cubes, the maximal shared cofactors between all cubes can be found by at most n^2 comparisons. Shared cofactors are stored by another tabular format, called `shared_cofactor_list`, which keeps the frequency of each shared cofactor and its dependent cubes.

Example 5.1: Consider a 3-input, 6-output LUT with equations $y_0 = x_0'x_1'x_2'+x_0'x_1x_2+x_0x_1'x_2$, $y_1 = x_0'x_1x_2+x_0x_1'x_2'$, $y_2 = x_0'x_1x_2' + x_0'x_1x_2 + x_0x_1x_2$, $y_3 = x_0'x_1x_2 + x_0x_1'x_2'$, $y_4 = x_0'x_2 + x_0x_2'$, and $y_5 = x_0'x_1x_2' + x_0x_1x_2$. It can be verified that this LUT has eight unique cubes with the `cube_list` shown in Table I. Table II illustrates the `shared_cofactor_list`.

Synthesis. To synthesize a (k, m)-LUT, we pick a shared cofactor from `shared_cofactor_list`. If the shared cofactor is also a cube in one of the outputs, it will be constructed on the respective output directly. Otherwise, a temporary output line that is not used at this step will be selected. However, if a cube should be constructed on all outputs, no temporary output is left for construction. In this case, an ancilla line is required. After constructing a shared cofactor on one output, all dependent cubes are constructed which leads to a one-time construction of the selected cofactor. Next, the dependent cubes and shared cofactors are removed from `cube_list` and `shared_cofactor_list`, respectively. This process is continued until no shared cofactor exists. Next, remaining cubes are constructed on respective outputs. To construct a shared cofactor, we always prefer to use one output with value 0 (Fig 4-a). However, we may not be able to find

such an *empty* line after applying several gates. In those cases, the idea of Fig. 4-b will be applied.

Lookahead. The order in which shared cofactors are processed affects the final circuit. To handle both the search space complexity and the quality of results, we use a *lookahead-based approach* with *depth* d. Accordingly, we start from a given function at level i and try all possible shared cofactors. This process is repeated for all resulting functions at level i, $i+1$, ..., $i+d$. Therefore, the algorithm explores at most N^d shared cofactors, if at most N shared cofactors exist at each level. Based on the achieved results at level $i+d$, the algorithm selects the best possible cofactor and backtracks to level i. Then, the algorithm applies the selected cofactor and repeats the same approach at level $i+1$.

The proposed LUT synthesis approach is shown in Algorithm 1. Lines 1-2 construct the required lists, lines 8-11 discuss synthesis, and lines 5-7 are related to lookahead.

Algorithm 1 LUT Synthesis

Input: A (k, m)-LUT with lookahead depth d.
Output: A quantum circuit that generates the LUT.

```
1: cube_list.construct();
2: shared_cofactor_list.construct();
3: cost = 0;
4: while ( !shared_cofactor_list.empty() ) do
5:     tree=construct_search_tree();
6:     min_cost_path=tree.exhaustive_search(d);
7:     cofactor f = min_cost_path.extract_first_node();
8:     f.implement_circuit();
9:     cost = cost + f.cost;
10:    cube_list.update(f);
11:    shared_cofactor_list.update(f);
12: end while
13: rCost = construct_remaining_cubes();
14: cost = cost + rCost;
```

VI. Experimental Results

We implemented the proposed LUT synthesis method in C++. To evaluate, we applied three different experiments.

- We compared our synthesis results with the systematic method in [7, Section 7.2] for those LUTs that appear in Shor's algorithm. These LUTs are the four costliest modular multiplications for semiprime M values with 9 bits or less in [7, Table 8]. The single-number cost model is used in both methods for comparison.
- We used the MCNC benchmarks from [17] and compared our results with the method in [13], which is one the most recent ESOP-based synthesis methods. Since the method in [13] reported quantum cost for their results, we included the quantum cost of our synthesized results for the MCNC benchmarks.
- Since we could not find relevant synthesized results for the binary welded tree in the literature, we synthesized oracle functions in Fig. 1 for black, red, green, and blue colors and applied the method in [14] implemented in [18] for the purpose of comparison.

We used EXORCISM-4 [17] to initially construct an ESOP representation for a given LUT and used it for synthesis. Furthermore, at most one ancilla is used in all circuits with a 3-level lookahead (i.e. $d = 3$). To control runtime, we limited the number of visited shared cofactors (i.e. N) at each level of lookahead. All experiments were done on an Intel Core i7-2600 machine with 8GB memory.

Table III shows the results of synthesizing LUTs for modular multiplications in Shor's algorithm. Besides the semiprime value M, for each method a triplet $(\mathcal{T}, \mathcal{C}, \texttt{cost})$ is reported, where $\mathcal{T}$ and $\mathcal{C}$ are the number of C^2NOT (Toffoli) and CNOT gates, respectively. The value `cost` is reported based on the single-number cost model [8]. On average, our proposed algorithm reduces the total cost by 52%. The synthesized circuit for $M = 65$ is shown in Figure 5. As shown, a post-synthesis optimization method may further improve the results.

To evaluate the proposed method in synthesizing irreversible functions, we used the MCNC benchmarks from [17] and compared our results with the results of [13]. Since in [13], quantum cost was used to calculate the synthesis cost, we used the same cost model. Synthesized results for the MCNC benchmarks are reported in Table IV. On average, our experiments show 28% improvement for the MCNC benchmarks.

We also examined the proposed approach in synthesizing the oracle functions of the binary welded tree in Fig. 1. Synthesis results for different oracle functions are reported in Table V. Quantum cost and the number of ancillae are compared. As can be seen, our method leads to more compact circuits with only one ancilla as compared to the method in [14].

VII. Conclusion

We addressed the problem of synthesizing a given LUT by reversible gates. Our algorithm is based on sharing possible cofactors and it tries different cofactors at each step with a lookahead to reduce cost. To construct cofactors on a limited number of qubits, the algorithm uses cofactor construction with un-computation. Our experiments showed the proposed method can significantly (52% on average) improve the synthesis cost of a recent method for those LUTs that appear in Shor's factoring algorithm. The results of applying the proposed method on the MCNC benchmarks show a considerable improvement in cost (28% on average) as compared with a recent ESOP-based method. We also showed that LUT synthesis can improve oracle of a binary welded tree.

Acknowledgments

This research was supported by the Intelligence Advanced Research Projects Activity (IARPA) via Department of Interior National Business Center contract number D11PC20165. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of IARPA, DoI/NBC, or the U.S. Government.

References

[1] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information*. Cambridge University Press, 2000.

TABLE III: Synthesis results for LUTs that appear in Shor's algorithm [7] for semiprime M values with 9 bits or less. For each method, the number of CNOT and Toffoli gates and cost are reported as a triplet $(\mathcal{T}, \mathcal{C}, \texttt{cost})$. Our synthesis algorithm improves the results in [7, Table 8] between 39.6% (M=253, marked with *) and 67.5% ($M = 217$, boldfaced). On average, the results in [7, Table 8] are improved by 52%. Gray cells include those cases that improvements are $< 45\%$. Runtime results are less than one minute in the proposed method. Both methods use at most one ancilla.

M	[7]	Ours	M	[7]	Ours	M	[7]	Ours	M	[7]	Ours	M	[7]	Ours
33	(49,7,252)	(16,30,110)	35	(51,7,262)	(20,31,131)	39	(44,4,224)	(16,33,113)	51	(27,4,139)	(14,12,82)	55	(47,9,244)	(18,35,125)
57	(51,6,261)	(14,30,100)	65	(41,12,217)	(15,21,96)	69	(50,7,257)	(20,48,148)	77	(55,6,281)	(24,35,155)	85	(36,2,182)	(12,19,79)
87	(56,9,289)	(20,59,159)	91	(56,6,286)	(15,45,120)	93	(50,3,253)	(15,32,107)	95	(43,9,224)	(15,54,129)	111	(51,7,262)	(14,34,104)
115	(45,11,236)	(20,41,141)	119	(57,6,291)	(15,48,123)	123	(61,6,311)	(15,58,133)	133	(50,14,264)	(10,38,88)	141	(57,8,293)	(19,43,138)
143	(49,10,255)	(20,41,141)	155	(62,11,321)	(18,52,142)	159	(52,13,273)	(18,44,134)	161	(58,11,301)	(15,51,126)	177	(48,8,248)	(17,56,141)
183	(67,11,346)	(19,66,161)	185	(61,7,312)	(17,50,135)	187	(70,9,359)	(17,72,157)	203	(63,12,327)	(19,69,164)	205	(40,3,203)	(11,31,86)
209	(60,12,312)	(17,61,146)	213	(63,13,328)	(20,80,180)	215	(62,13,323)	(17,33,118)	**217**	**(39,5,200)**	**(9,20,65)**	219	(53,9,274)	(14,46,116)
221	(60,9,309)	(15,42,117)	235	(56,16,296)	(20,55,155)	237	(62,10,320)	(20,68,168)	247	(51,11,266)	(14,58,128)	253*	(47,12,247)	(20,49,149)
259	(47,12,247)	(14,52,122)	267	(62,7,317)	(17,55,140)	287	(63,17,332)	(20,61,161)	291	(58,16,306)	(15,56,131)	295	(76,17,397)	(23,95,210)
299	(56,12,292)	(15,72,147)	301	(65,8,333)	(22,80,190)	303	(54,5,275)	(18,74,164)	305	(59,9,304)	(19,66,161)	309	(59,17,312)	(19,71,166)
319	(65,13,338)	(20,74,174)	323	(74,12,382)	(14,73,143)	327	(62,11,321)	(15,61,136)	329	(59,13,308)	(18,75,165)	335	(54,11,281)	(19,67,162)
339	(67,8,343)	(18,63,153)	341	(61,5,310)	(15,43,118)	355	(75,13,388)	(21,74,179)	365	(62,10,320)	(14,63,133)	371	(61,13,318)	(19,88,183)
377	(70,10,360)	(19,73,168)	381	(56,7,287)	(21,33,138)	391	(70,12,362)	(17,59,144)	393	(61,20,325)	(18,76,166)	395	(63,14,329)	(17,85,170)
403	(72,9,369)	(20,75,175)	407	(52,10,270)	(20,58,158)	411	(64,9,329)	(19,65,160)	413	(71,11,366)	(21,73,178)	415	(58,14,304)	(22,57,167)
417	(66,16,346)	(17,76,161)	427	(71,11,366)	(18,97,187)	437	(61,15,320)	(19,83,178)	445	(65,10,335)	(22,51,161)	447	(60,14,314)	(20,61,161)
451	(68,9,349)	(15,80,155)	453	(63,12,327)	(20,79,179)	469	(58,16,306)	(18,71,161)	471	(82,8,418)	(19,72,167)	473	(69,18,363)	(18,72,162)
481	(64,13,333)	(15,47,122)	485	(74,9,379)	(15,68,143)	493	(64,14,334)	(19,46,141)	497	(61,15,320)	(19,71,166)	501	(62,16,326)	(20,75,175)
511	(54,6,276)	(14,39,109)												

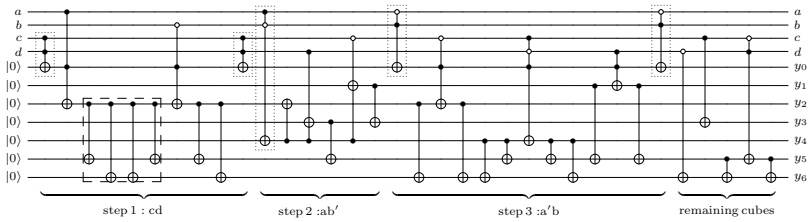

Fig. 5: The result of applying the proposed synthesis algorithm to synthesize the $(4, 7)$-LUT in Shor's algorithm for $M = 65$. The ESOP expansion for outputs can be represented as $y_0 = 0$, $y_1 = ab'c' \oplus a'bd$, $y_2 = ab' \oplus a'bc' \oplus acd \oplus b'cd$, $y_3 = ab'd \oplus ab'c' \oplus c$, $y_4 = ab' \oplus a'bcd'$, $y_5 = ab'd \oplus a'bcd' \oplus acd \oplus b'cd \oplus a'bd \oplus c'd$, $y_6 = d' \oplus a'bc' \oplus a'bcd' \oplus acd \oplus b'cd \oplus c'd$. Shared cofactors are highlighted with dotted boxes. As shown in the dashed box, a post-synthesis optimization can further improve the circuit.

TABLE IV: Synthesis results for the MCNC Benchmarks. On average, the results of [13] are improved by 28%. Runtime results vary from a few seconds for small functions to about 5 minutes for large functions. Our method uses at most one ancilla.

Circuit	[13]	Our Method	Imp. (%)	Circuit	[13]	Our Method	Imp. (%)	Circuit	[13]	Our Method	Imp. (%)	Circuit	[13]	Our Method	Imp. (%)	Circuit	[13]	Our Method	Imp. (%)
5xp1	786	576	27	9symml	10943	3068	72	alu4	41127	33191	19	apex4	35840	28313	21	apex5	33830	20935	38
apla	1683	1026	39	bw	637	616	3	cordic	187620	90100	52	C7552	399	253	37	clip	3824	2657	31
cm42a	161	120	25	cu	781	365	53	dc1	127	109	14	dc2	1084	736	32	decod	399	253	37
dist	3700	2412	35	dk17	1014	623	39	ex1010	52788	43543	18	ex5p	3547	2566	28	f2	112	84	25
f51m	28382	23212	18	frg2	112008	97837	13	ham7	67	65	3	hwb8	8195	6108	25	in0	7949	6885	13
inc	892	624	30	misex1	332	218	34	misex3c	49720	42349	15	misex3	49076	40470	18	mlp4	2496	1784	29
pdc	30962	27098	12	root	1811	1210	33	sao2	3767	1484	61	seq	33991	23034	32	sqr6	583	494	15
urf3	53157	45014	15	wim	139	97	30	z4ml	489	402	18								

TABLE V: Results for the oracles in Fig. 1. For [14], we used m CNOTs to initially copy inputs to outputs to keep inputs unchanged.

Color	(cost, #ancillae)		
	[14]	Ours	Imp.(%)
Blue	(339,24)	(226,1)	(33,96)
Red	(268,21)	(256,1)	(4,95)
Green	(298,23)	(274,1)	(8,96)
Black	(213,19)	(188,1)	(11,95)

[2] D. Bacon and W. van Dam, "Recent progress in quantum algorithms," *Commun. ACM*, vol. 53, pp. 84–93, Feb. 2010.

[3] L. M. K. Vandersypen *et al.*, "Experimental realization of an order-finding algorithm with an NMR quantum computer," *Phys. Rev. Lett.*, vol. 85, pp. 5452–5455, Dec. 2000.

[4] N. Xu *et al.*, "Quantum factorization of 143 on a dipolar-coupling nuclear magnetic resonance system," *Phys. Rev. Lett.*, vol. 108, p. 130501, Mar. 2012.

[5] G. Schaller and R. Schützhold, "The role of symmetries in adiabatic quantum algorithms," *Quantum Info. Comput.*, vol. 10, no. 1, pp. 109–140, Jan. 2010.

[6] A. M. Childs *et al.*, "Exponential algorithmic speedup by a quantum walk," in *Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*, pp. 59–68, 2003.

[7] I. L. Markov and M. Saeedi, "Constant-optimized quantum circuits for modular multiplication and exponentiation," *Quantum Info. Comput.*, vol. 12, no. 5-6, pp. 361–394, May 2012.

[8] D. Maslov and M. Saeedi, "Reversible circuit optimization via leaving the Boolean domain," *IEEE Trans. CAD*, vol. 30, no. 6, pp. 806 – 816, Jun. 2011.

[9] M. Saeedi and I. L. Markov, "Synthesis and optimization of reversible circuits - a survey," *ACM Computing Surveys, arXiv:1110.2574*, 2012.

[10] C.-F. Chiang, D. Nagaj, and P. Wocjan, "Efficient circuits for quantum walks," *Quantum Info. Comput.*, vol. 10, no. 5, pp. 420–434, May 2010.

[11] M. Saeedi *et al.*, "Reversible circuit synthesis using a cycle-based approach," *J. Emerg. Technol. Comput. Sys.*, vol. 6, no. 4, pp. 13:1–13:26, Dec. 2010.

[12] K. Fazel, M. Thornton, and J. Rice, "ESOP-based Toffoli gate cascade generation," in *Communications, Computers and Signal Processing, IEEE Pacific Rim Conference on*, pp. 206 –209, Aug. 2007.

[13] N. M. Nayeem and J. E. Rice, "A shared-cube approach to ESOP-based synthesis of reversible logic," in *Facta universitatis - series: Electronics and Energetics*, vol. 24, no. 3, pp. 385–402, Dec. 2011.

[14] R. Wille and R. Drechsler, "BDD-based synthesis of reversible logic for large functions," in *Design Automation Conference*, pp. 270–275, 2009.

[15] M. Plesch and Č. Brukner, "Quantum-state preparation with universal gate decompositions," *Phys. Rev. A*, vol. 83, p. 032302, Mar 2011.

[16] R. K. Brayton *et al.*, *Logic Minimization Algorithms for VLSI Synthesis*. Kluwer Academic Publishers, 1984.

[17] A. Mishchenko and M. Perkowski, "Fast heuristic minimization of exclusive sum-of-products," in *Reed-Muller Workshop*, 2001.

[18] M. Soeken *et al.*, "RevKit: A toolkit for reversible circuit design," *Workshop on Reversible Computation*, 2010.