A Library-Based Synthesis Approach for Reversible Logic

Mehdi Saeedi, Mehdi Sedighi, and Morteza Saheb Zamani
Quantum Design Automation Group, Computer Engineering Department,
Amirkabir University of Technology
Tehran, Iran
Email: {msaeedi, msedighi, szamani}@aut.ac.ir

***Abstract*—Synthesis of reversible logic has received significant attention in the recent years and many synthesis approaches for reversible logic have been proposed so far. In this paper, a library-based synthesis approach for reversible circuits is proposed where an input specification is considered as a permutation comprising a set of cycles. In order to synthesize a given permutation, a library which contains seven building blocks is used where each building block is a cycle of length less than 6. We also propose a decomposition algorithm which produces all possible minimal and inequivalent factorizations for a given cycle of length greater than 5. All decompositions contain the maximum number of disjoint cycles.**

The generated decompositions are used in conjunction with a cycle assignment algorithm which is proposed based on the graph matching problem to select the best possible cycle pairs. Then, each pair is synthesized by using the available components of the library. The decomposition algorithm and the cycle assignment method are considered as a binding method which selects a building block from the given library for each cycle. Experimental results reveal that the proposed library-based approach can produce low-cost circuits compared with the available synthesis methods.

I. Introduction

In 1961, Landauer proved that using conventional logic gates leads to a certain amount of energy dissipation per irreversible bit operation regardless of the underlying technology [1]. In 1973, Bennett stated that to avoid power dissipation in a circuit, it must be built from reversible gates [2]. Currently, reversible computing has received considerable attention in particular in low-power CMOS design (e.g., see [3]).

Quantum gates are inherently reversible [4]. Thus, reversible logic has also found great interest in the domain of quantum computation. As such, various Boolean reversible gates are used in different quantum algorithms [5]. In fact, constructing efficient circuits with Boolean reversible gates is considered an important step towards realization of quantum systems [6].

Reversible logic synthesis requires a systematic method to produce efficient circuits. To address this need, several synthesis algorithms for reversible functions have been proposed where both exact (e.g., [7]–[10]) and heuristic approaches (e.g., [11]–[14]) were applied.

While exact reversible synthesis algorithms are useful to obtain optimal circuits for small functions, they cannot be used to handle relatively large specifications due to the exponential search space growth. On the other hand, among the available heuristic methods, those which reduce the number of elementary gates with the penalty of using other resources may have major problems for quantum logic. For example, since number of qubits is very restricted in quantum computation, approaches that use an arbitrary number of garbage lines (e.g., [13] which was proposed for reversible circuits) cannot be applied to quantum logic.

In [15], a synthesis algorithm has been proposed that considers reversible functions as a set of cycles where each cycle was implemented by several reversible gates. By extending the results of [15], in this paper a library-based synthesis algorithm for reversible circuits is proposed where a set of building blocks and a binding algorithm are introduced to be used in a unified library-based synthesis approach.

The rest of the paper is organized as follows. Basic concepts are introduced in Section II. The synthesis algorithm of [15] is described in Section III. The proposed library-based synthesis method is introduced in Section IV. Experimental results are shown in Section V and finally, Section VII concludes the paper.

II. Basic Concepts

A. Reversible Logic

Let A be any set and define $f : A \rightarrow A$ as a one-to-one and onto transition function. The function f is called a *permutation function*, as applying f to A leads to a set with the same elements of A and probably in a different order. If $A = \{1, 2, 3, \ldots, m\}$, there exist two elements a_i and a_j belonging to A such that $f(a_i) = a_j$. A *k-cycle* with *length* k is denoted as $(a_1, a_2, \ldots, a_k)$ which means that $f(a_1) = a_2$, $f(a_2) = a_3$, ..., and $f(a_k) = a_1$. A given k-cycle $(a_1, a_2, \ldots, a_k)$ could be written in many different ways such as $(a_2, a_3, \ldots, a_k, a_1)$.

Cycles c_1 and c_2 are called *disjoint* if they have no common members, i.e., $\forall a_i \in c_1, a_i \notin c_2$. Any permutation can be written uniquely, except for the order, as a product of disjoint cycles. The unique cycle form of a permutation is called *canonical cycle form* (*CCF*) [15]. If two cycles c_1 and c_2 are disjoint, they can commute, i.e., $c_1 c_2 = c_2 c_1$.

An n-input, n-output, fully specified Boolean function is reversible if it maps each input pattern to a unique output pattern. Each reversible function can be considered as a permutation function. A *gate* is called reversible if it

realizes a reversible function. A *generalized Toffoli gate* $C^mNOT(x_1, x_2, \cdots, x_{m+1})$ passes the first m lines unchanged. These lines are referred as control lines. This gate flips the $(m+1)^{th}$ line if and only if the control lines are all one. For $m = 0$ and $m = 1$, the gates are called *NOT* and *CNOT*, respectively. For $m = 2$, the gate is called C^2NOT or *Toffoli*.

Outputs that are not required in the function specification are called *garbage* bits. The number of elementary gates required for simulating a given gate is called *quantum cost*.

Positive polarity Reed-Muller (*PPRM*) expansion can be used to describe a reversible specification. PPRM expansion uses only un-complemented (or positive) variables and it can be derived from the EXOR-Sum-of-Products (*ESOP*) description by replacing a' with $a \oplus 1$ for a complemented variable a. In addition, some algebraic manipulation of product terms may also be done to simplify the equations. The PPRM expansion of a function is canonical and is defined as (1).

$$f(x_1, x_1, ..., x_n) = a_0 \oplus a_1x_1 \oplus \cdots \oplus a_nx_n \oplus a_{12}x_1x_2 \oplus \cdots \oplus a_{n,n-1}x_{n-1}x_n \oplus \ldots \oplus a_{12\ldots n}x_1x_2\cdots x_n \quad (1)$$

B. Cycle Factorization

A reversible specification can be considered as a permutation function which includes a set of cycles of various lengths. Let $\sigma_1, \cdots, \sigma_m$ be a factorization of the cycle $(a_1, a_2, \ldots, a_n)$ into a product of smaller cycles. We say the factorization is of *type* $\alpha = (\alpha_2, \ldots, \alpha_k)$ if among $\sigma_j (1 \leq j \leq m)$ there are exactly α_2 2-cycles, α_3 3-cycles and so on. Let us define

$$\langle\alpha\rangle = \Sigma_{j\geq 2}(j-1)\alpha_j \quad (2)$$

where α satisfies $\langle\alpha\rangle \geq n-1$. For the case of equality, the factorization is called *minimal*. Two cycle factorizations are called *equivalent* if one can be obtained from the other by repeatedly exchanging adjacent factors that are disjoint.

***Example** 1:* Consider a given cycle π=(a, b, c, d, e) of length $n = 5$. It can be verified that π can be factorized into (a, b) (a, c) (a, d, e) with the cycle type (2, 1). Note that cycles are applied from left to right. For this factorization, we have $\langle\alpha\rangle$=$1*2+2*1 = 4$. Since $\langle\alpha\rangle = n-1$ this factorization is minimal.

Cycle factorization is used to extract library elements from a given specification as discussed in Section IV in detail.

C. Graph Matching

In order to select library elements in the proposed library-based synthesis algorithm, the graph perfect matching algorithm is applied. Given a graph $G = (V, E)$, a *matching* M in G is a set of pairwise non-adjacent edges; that is, no two edges share a common vertex. A vertex is *matched* if it is incident to an edge in the matching. Otherwise the vertex is *unmatched*. A *maximum matching* is a matching that contains the largest possible number of edges. There may be many maximum matchings. The *matching number* of a graph is the size of a maximum matching.

A *perfect matching* is a matching which matches all vertices of the graph. That is, every vertex of the graph is incident to exactly one edge of the matching. In a weighted bipartite graph, each edge has an associated value. A *minimum weighted bipartite matching* is defined as a perfect matching where the sum of the values of the edges in the matching has a minimal value. If the graph is not complete, missing edges are inserted with value zero.

III. Previous Work

The authors of [6] proposed a NCT-based synthesis method which applies NOT (N), Toffoli (T), CNOT (C) and Toffoli (T) gates in order (i.e.,, the T|C|T|N method) to synthesize a given permutation. In the first C|T|N part, the terms 0 and 2^i of a given reversible function are positioned at their right locations while the last Toffoli network places the other truth table terms in their right positions. For the last Toffoli part, a given k-cycle is decomposed into a set of transpositions. Subsequently, each pair of disjoint transpositions (a, b) (c, d), is implemented by a synthesis algorithm that maps a, b, c and d to 2^n-4, 2^n-3, 2^n-2 and 2^n-1, respectively, i.e.,, the π circuit. Then, the permutation (2^n-4, 2^n-3) (2^n-2, 2^n-1) is implemented by a circuit called κ_0. Finally, the reverse π circuit, i.e.,, π^{-1}, is applied to transform 2^n-4, 2^n-3, 2^n-2 and 2^n-1 into a, b, c and d, respectively. It can be verified that the $\pi\kappa_0\pi^{-1}$ circuit implements the permutation (a, b) (c, d). An extension of [6] was suggested in [16] which produces better quantum cost by applying the unit-cost NOT and CNOT gates instead of using Toffoli gates with cost 5 in many situations.

In [6], for the last Toffoli part, a given k-cycle is decomposed into a set of transpositions. Subsequently, each pair of disjoint transpositions (a, b) (c, d), is implemented by a synthesis algorithm that maps a, b, c and d to 2^n-4, 2^n-3, 2^n-2 and 2^n-1, respectively, i.e.,, the π circuit. Then, the permutation (2^n-4, 2^n-3) (2^n-2, 2^n-1) is implemented by a circuit called κ_0. Finally, the reverse π circuit, i.e.,, π^{-1}, is applied to transform 2^n-4, 2^n-3, 2^n-2 and 2^n-1 into a, b, c and d, respectively. It can be verified that the $\pi\kappa_0\pi^{-1}$ circuit implements the permutation (a, b) (c, d). An extension of [6] was suggested in [16] which produces better quantum cost by applying the unit-cost NOT and CNOT gates instead of using Toffoli gates with cost 5 in many situations. In our previous work [15], a cycle-based synthesis algorithm has been proposed where cycles of lengths less than 4 are synthesized directly. More exactly, in [15] a set of synthesis algorithms have been proposed to synthesize a pair of 2-cycles, a single 3-cycle, and a pair of 3-cycles. Each cycle has been called a *building block* or an *elementary cycle*. In order to improve the synthesis cost, the authors extended the building blocks to include a single 4-cycle followed by a single 4-cycle or a single 2-cycle, a single 5-cycle and a pair of 5-cycles. In addition, we used NOT and CNOT gates (with unit cost) instead of Toffoli (with cost 5) in many situations.

As an example, a pair of 2-cycles $(a, b)(c, d)$ is synthesized as follows. First the terms $a, b, c,$ and d are mapped into 4, 1, 2 and $2^{(n-1)}+3$, respectively. Then a pre-designed circuit

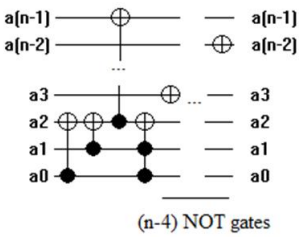

Fig. 1. (2,2) synthesis algorithm as proposed in [15]. NOT and CNOT gates were used instead of Toffoli to reduce quantum cost.

Fig. 2. The κ_0 circuit [15]

such as the one shown in Fig. 1, is applied followed by the circuit shown in Fig. 2 (namely κ_0). Afterwards, the gates applied before the κ_0 circuit are applied in the reverse order. For example, Fig. 3 illustrates the synthesized circuit for the permutation (112, 119)(113, 118).

On the other hand, to synthesize a given *large* cycle of length k ($k > 3$) the authors used one possible decomposition to extract the suggested building blocks (i.e., cycles) from the input specification (i.e., permutation) that leads to a set of cycles of lengths 3 and probably a cycle of length less than 3. Since we used an extended set of building blocks, the decomposition algorithm was modified to detach 5-cycles. Therefore, the results of the decomposition algorithm is a set of cycles of lengths 5 and probably a cycle of length less than 5. For example, consider π=(3, 5, 6, 7, 9, 10, 11, 12, 13, 14, 15, 17, 18, 19, 20, 21) (22, 23, 24, 25, 26, 27) (28, 29) (30, 31) where detaching all 5-cycles leads to π=(3, 5, 6, 7, 9) (10, 11, 12, 13, 14) (15, 17, 18, 19, 20) (22, 23, 24, 25, 26) (21, 3, 10, 15) (22, 27) (28, 29)(30, 31).

Besides the applied synthesis scheme, different synthesis algorithms used different representations for their input specifications. Among the available models, truth table (e.g., see [11], [12], [17], [18]) and positive polarity Reed-Muller (PPRM) expansions (e.g., see [14], [19]) have been widely used. The selected model works as an *intermediate format* (IF) for the respective synthesis algorithm and is placed between two levels of abstraction (i.e., input specification and gate-level circuit). In this paper, CCF representation has been selected as an IF as discussed in Section IV-A. Note that CCF has been used to describe the input specification in [15] and [6] to some extent.

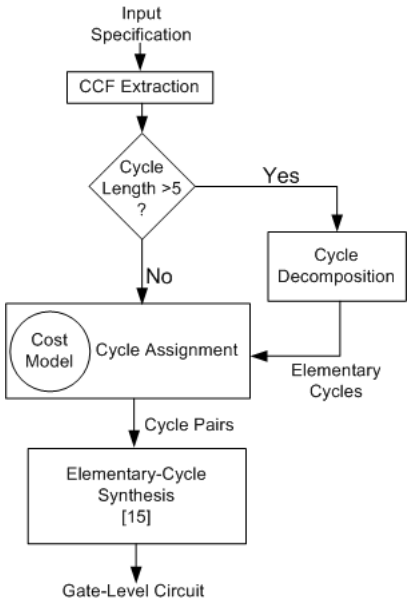

Fig. 4. Proposed synthesis approach

In the next section, we propose a library-based synthesis approach for reversible circuits based on the results of [15].

IV. Proposed Approach

Fig. 4 shows the proposed approach. A CCF representation is extracted from the input specification first (Section IV-A) and then is gradually mapped into a reversible circuit. If cycle length is greater than 5, we apply a cycle decomposition algorithm (Section IV-B) to construct elementary cycles. Next, a cycle assignment method (Section IV-C) is applied to construct cycle pairs based on the well-known graph matching problem. Then, each pair is synthesized by applying the method of [15].

A. Intermediate Format

Compared with the truth table model, CCF removes fixed rows of a given truth table and hence are very efficient for large functions. In other words, a reversible function with n inputs and n outputs needs a truth table of size $n \times 2^n$ where each row may or may not be changed depending on the specification. On the other hand, a synthesis algorithm for reversible circuits returns the changed rows to their right positions by applying a set of reversible gates. Fixed rows can be removed to save space if the synthesis algorithm does not use it in the synthesis step directly. In [11], the authors reported that the applicability of their synthesis algorithm was limited due to the memory constraint occurred during the representation of large input specification in the truth table format.

Moreover, while some truth table-based approaches likes those introduced in [11] and [18] considered both input-to-output and output-to-input transformations at the same time (namely *bidirectional method*), the mentioned transformations have equal CCF representations. Therefore, there is no need to consider both transformations at the synthesis step. Hence, lower complexities should be handled by the synthesis method.

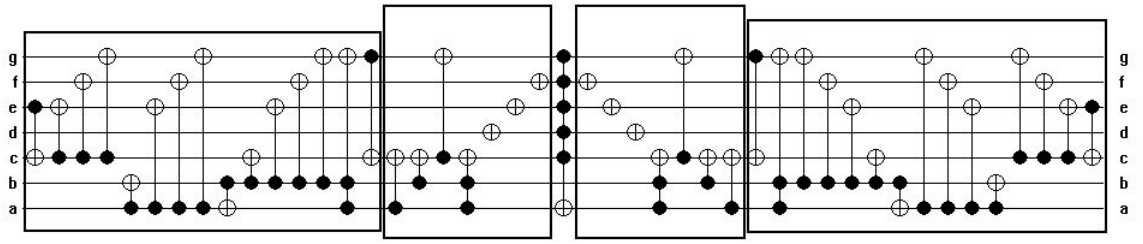

Fig. 3. A sample circuit synthesized by the method of [15]. NOT and CNOT gates were used instead of Toffoli to reduce quantum cost.

TABLE I
AVERAGE DISTRIBUTION OF CYCLE LENGTHS FOR AVAILABLE BENCHMARK FUNCTIONS

2-cycle	3-cycle	4-cycle	5-cycle	$\geq$ 6-cycle
18.95%	6.57%	8.79%	2.87%	62.82%

On the other hand, for a given n-input/n-output reversible function, n PPRM expansions can be extracted which removes explicit values of truth table rows. Of course, the truth table rows can be recovered from the PPRM expansions with cost. While PPRM notation received attractions in some synthesis algorithms, it cannot be used in the proposed method since explicit row values are needed in this paper. Altogether, CCF benefits from compact notation of PPRM expansions (particularly for large specifications where many input combinations map into themselves) and explicit values of truth table representation. Therefore, CCF is used as the selected IF in the proposed synthesis approach.

Having an input specification in the CCF format, the next step is to synthesize it according to [15] where small cycles are synthesized by the suggested building blocks directly. Table I shows the average distribution of cycle lengths for the benchmark functions [20]. As shown in this table, more than 60% of cycle lengths are greater than 5. Therefore, many cycles should be decomposed into the proposed set of cycles and hence, cycle decomposition can affect the synthesis costs considerably. In the following, the effects of cycle decomposition on the synthesis results are examined.

B. Cycle Decomposition

Since each decomposed cycle should be synthesized by a set of reversible gates, reducing the number of decomposed cycles is always preferred in [15] to reduce final cost. Moreover, as discussed in Section III, all decomposed disjoint cycles can commute to find the best possible selection of cycle pairs for having lower synthesis cost. Therefore, each large cycle should be synthesized so that the minimal number of inequivalent 5-cycles are generated. Besides, number of disjoint 5-cycles should be maximized.

Based on the above considerations, to decompose a given large cycle of length n into a set of 5-cycles, we impose the following conditions:

- All decomposed cycles should be of length 5 except at most one cycle which is of length less than 5.
- The cycle factorization should be minimal.
- Inequivalent cycle factorization is considered.
- Maximum number of disjoint cycles should be produced.

Consider a cycle π of length n. The maximum number of disjoint cycles resulted from an inequivalent 5-cycle factorization is $\lfloor n/5 \rfloor$. In addition, assume that all disjoint 5-cycles are detached. According to the minimal factorization together with Equation (2), we have $4 \times \lfloor n/5 \rfloor + (L-1) = n-1$ where L is the length of resulted non-disjoint cycle after detaching all disjoint 5-cycles (denoted as $\acute{\pi}$ in the following). It can be verified that L is equal to $n-4\times\lfloor n/5 \rfloor$. Note that $\acute{\pi}$ includes at most four elements of π which does not belong to the detached 5-cycles. In addition, it has $\lfloor n/5 \rfloor$ elements of π each of which belongs to exactly one disjoint cycle inserted to recover the original cycle π from the set of disjoint 5-cycles. Considering the minimal length of $\acute{\pi}$, exactly one element exists in $\acute{\pi}$ for each disjoint 5-cycle.

In order to have *both* the minimum number of decomposed cycles and the maximum number of disjoint cycles for a given cycle π, the order of elements in each disjoint cycle should be the same as the original cycle π; otherwise some extra cycles should be inserted to construct the given permutation. Consider the following example for more detail:

***Example** 2:* Suppose a cycle π = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19) of length $n = 19$. Since $n = 19$, $\lfloor n/5 \rfloor = 3$. The decomposition π = (1,2,3,4,5) (6,7,8,9,10) (11,12,13,14,15) (16,17,18,19,1) (6,11,16) meets the conditions and it is minimal. On the other hand, if the order of elements in disjoint cycles are modified, several extra cycles should be included. As an example, the decomposition π = (1,2,4,5,6) (3,7,8,9,10) (11,12,13,14,15) (16,17,18,19,1) (3,11,16) (4,3,7) is not minimal.

To have the minimum number of decomposed 5-cycles, the ordering of those elements which do not belong to any disjoint cycle should be the same as the original large cycle too; otherwise some extra cycles should be inserted to fix the locations of those elements. Altogether, for a cycle $\pi = (a_1, a_2..., a_n)$ of length $n > 5$, there are $N_{DCM}(n) = L^{(0)} \times L^{(1)} \times \ldots L^{(i)}$ factorizations each of which has the required four conditions discussed above and we have $L^{(0)} = n$, $L^{(i)} = L^{(i-1)} - 4 \times \lfloor L^{(i-1)}/5 \rfloor$, $L^{(i+1)} < 5$.

There are many ways of decomposing a given large cycle into a set of cycles of length less than 6 with minimum number of decomposed cycles and maximum number of

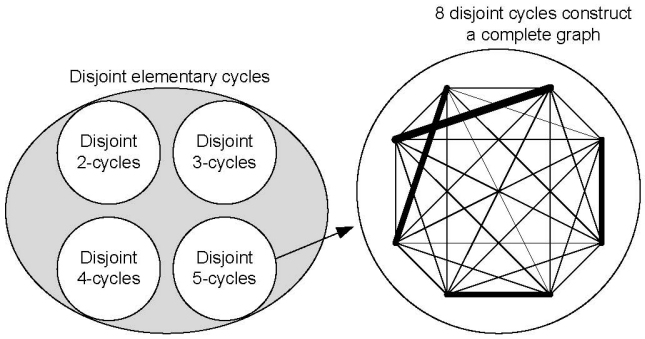

Fig. 5. Cycle assignment. Different nodes represent different disjoint cycles. A connected edge between two nodes denotes the probability of synthesizing the cycles as a pair. Each edge contains a weight which is the synthesis cost of the involved cycles.

disjoint cycles. In the next subsection, the process of selecting cycle pairs are evaluated. In Section V, experimental results are discussed.

C. Cycle Assignment

For a given cycle π of length n, $N_{DCM}(n)$ different decompositions are possible where each decomposition includes $\lfloor n/5 \rfloor$ disjoint 5-cycles. Non-disjoint cycles cannot be arbitrarily moved.

In order to find cycle pairs, we model the cycle assignment step as a graph perfect matching problem. For a set with N elementary cycles, $N \times (N-1)/2$ cycle pairs can be determined where each pair can be synthesized by a specific cost. Since each cycle pair can be considered as a valid cycle assignment, we first synthesize each cycle pair using the method of [15]. Then, a weighted graph is constructed with N nodes and $N \times (N-1)/2$ edges. The synthesis cost for each cycle pair is used as the weight of an edge between the respective nodes. Next, a graph perfect matching algorithm is applied to find the best possible matching with minimum cost. Therefore, cycle assignments which produce lower total cost are found.

Fig. 5 illustrates the cycle assignment problem for the generated disjoint 5-cycles. As can be seen in this figure, 8 disjoint 5-cycles exist which construct a complete graph on 8 nodes. A possible cycle assignment is shown with solid edges. It is worth noting that since all cycles of a given input specification are disjoint, the resulted set of 2-cycles contains only disjoint cycles. Therefore, it can be possible to apply cycle assignment step to the elementary 2-cycles too. Similarly, this process can be repeated for all 3-cycles and 4-cycles.

V. Experimental Results

The proposed library-based synthesis approach was implemented in C++ and all of the experiments were done on an Intel Pentium IV 2.2GHz computer with 2GB memory. In order to find a perfect matching on a given graph, we used Blossom V implementation [21]. In addition, we used one of the most recent synthesis tools proposed in [11] for our comparisons.

In all experiments, the post-processing algorithm proposed in [16] was applied to simplify circuits produced by our synthesis approach. In addition, the synthesis algorithm of [11] was applied in '*synthesized/ resynthesized using 3 methods*' mode for circuits with $n < 15$ and in '*synth/resynth with MMD (15+ variables)*' for $n > 15$. For the synthesis algorithm of [11], the synthesis algorithm, the templates matching method, the random and exhaustive driver algorithms were applied sequentially to synthesize each function with a time limit of 12 hours as done in [11]. Bidirectional and quantum cost reduction modes were also applied.

To evaluate the proposed synthesis approach, the completely specified reversible benchmark functions with more than six variables [20] were examined as done in [15]. We first fixed zero and 2^i terms by applying a few Toffoli and CNOT gates using the method of [6]. Then, our synthesis approach was applied. To compare the results, we evaluated all synthesis algorithms in terms of quantum cost and the number of garbage bits. Quantum costs were calculated based on [17].

The results of our synthesis algorithm and the previous best-proposed circuits are reported in Table II. Headings 'w/ g' and 'w/o g' stand for 'with garbage' and 'without garbage', respectively. In addition, '# g' denotes the number of garbage line. The symbol '-' is used if the algorithm fails to synthesize the circuit.

The function urf6 has 15 variables and the synthesis tool of [11] was applied in 'synth/resynth with MMD (15+ variables)' mode. However, the tool failed to synthesize the function. Then, the tool was applied in 'synthesized/resynthesized using 3 methods' mode. The synthesis tool again failed to synthesize the circuit after 12 hours. Also, for urf4, the synthesis algorithm of [11] failed to finish. For urf1, urf2, urf3, and urf5 functions, several circuits were reported in [20] that were synthesized using the method of [18]. The resulted costs for these circuits are 45855, 16152, 121716, and 24253, respectively. Since applying the method of [11] significantly improves the previous costs, we reported the new ones in Table II.

Since the number of valid decompositions for each cycle of the reversible benchmarks grows rapidly, it is not possible to evaluate all decompositions to select the best possible one for each cycle. Therefore, we limited the runtime to 30 minutes and generated a few decompositions for each cycle randomly. Next the generated decompositions were evaluated to find the best possible cost.

Table II shows the results. In this table, the synthesis results of applying the method of [15] for only one decomposition and with a trivial cycle assignment where consecutive cycles are assigned to each other are shown (1-Way DCM+CA). As can be seen in Table II, the proposed library-based synthesis algorithm can lead to lower synthesis costs for almost all benchmark functions. Compared with [11], the approach uses no garbage bit and synthesizes the specifications very fast (at most in 30 minutes for each circuit). According to [6], even

TABLE II
THE COMPARISON COSTS OF OUR LIBRARY-BASED SYNTHESIS APPROACH WITH THE ALGORITHM OF [11]. IMPROVED RESULTS ARE IN BOLD. FOR [11], A TIME LIMIT OF 12 HOURS WAS APPLIED AS DONE IN [11]. AT MOST 30 MINUTES WERE REQUIRED FOR EACH CIRCUIT IN THE PROPOSED METHODOLOGY.

Benchmark Function	n	[11] #g	[11] w/ g	[11] w/o g	1-Way (w/o g) DCM+CA	ours w/o g
ham7	7	0	**49**	49+	2117	1804
hwb7	7	1	**2609**	2613	3177	2727
hwb8	8	1	6197	7015	7163	**6535**
hwb9	9	1	20378	22510	16283	**15462**
hwb10	10	1	46597	59197	36182	**34224**
hwb11	11	1	122144	136760	91973	**86942**
urf1	9	1	21850	23983	17281	**16619**
urf2	8	1	8161	9418	7291	**6600**
urf3	10	1	49843	61046	38133	**36927**
urf4	11	-	-	-	93992	**90696**
urf5	9	1	12782	14225	14876	**13930**
urf6	15	-	-	-	17367	**16687**

permutations do not require garbage bits for the implementation by NOT, CNOT and Toffoli gates. Hence. the proposed approach are optimal in terms of using garbage bits.

VI. FUTURE DIRECTIONS

While the proposed approach needs much less time to synthesize a given specification in contrast to the existing methods, runtime can be improved more. In other words, since we used few decompositions at each step (due to the exponential search space), there is no need to construct all valid decompositions in the synthesis stage. This optimization can improve both memory usage and runtime significantly. In addition, while selecting a number of decompositions randomly reveals the benefits of the proposed approach considerably, it may be possible to guide the decomposition selection algorithm according to a scenario. This can lead to better runtime and synthesis costs. Working on some parts of these improvements is under way.

VII. CONCLUSION

Reversible logic has attracted great attention recently due to its applications in low-power CMOS design and quantum computation. In this paper, a synthesis approach for reversible circuits was proposed which used a set of building blocks and a library to synthesize a given specification. To this end, each input specification is considered as a permutation with several cycles where each cycle is synthesized by some reversible gates. If a given cycle is found in the library, it is synthesized directly; otherwise, the proposed decomposition algorithm detached the building blocks from the given cycle. The decomposition algorithm explores all possible minimal and inequivalent factorizations where the number of disjoint cycles is maximized. Since to synthesize a given permutation, cycle pairs should be selected to reduce synthesis cost, a cycle assignment algorithm was proposed based on the graph perfect matching algorithm too. Experimental results on reversible functions illustrated the advantage of the proposed approach in reducing both synthesis cost (i.e., quantum cost and number of garbage lines) and runtime.

ACKNOWLEDGMENT

We would like to acknowledge Dmitri Maslov from University of Waterloo for providing an executable version of his synthesis tool.

REFERENCES

[1] R. Landauer. Irreversibility and heat generation in the computing process. *IBM Journal of Research and Development*, 5:183–191, July 1961.

[2] C. Bennett. Logical reversibility of computation. *IBM Journal*, 17(6):525–532, November 1973.

[3] G. Schrom. *Ultra-Low-Power CMOS Technology*. PhD thesis, Technischen Universitat Wien, June 1998.

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

[5] A. Barenco et al. Elementary gates for quantum computation. *APS Physical Review A*, 52:3457–3467, 1995.

[6] V. V. Shende, A. K. Prasad, I. L. Markov, and J. P. Hayes. Synthesis of reversible logic circuits. *IEEE Trans. on CAD*, 22(6):710–722, June 2003.

[7] W. N. N. Hung, X. Song, G. Yang, J. Yang, and M. Perkowski. Optimal synthesis of multiple output boolean functions using a set of quantum gates by symbolic reachability analysis. *IEEE Trans. on CAD*, 25(9):1652–1663, September 2006.

[8] D. Große, R. Wille, G. W. Dueck, and R. Drechsler. Exact synthesis of elementary quantum gate circuits for reversible functions with don't cares. *International Symposium on Multiple Valued Logic*, pages 214–219, 2008.

[9] D. Große, R. Wille, G.W. Dueck, and R. Drechsler. Exact multiple control toffoli network synthesis with SAT techniques. *IEEE Trans. on CAD*, 28(5):703–715, 2009.

[10] R. Wille, H. M. Le, G. W. Dueck, and D. Große. Quantified synthesis of reversible logic. *Design, Sutomation and Test in Europe*, pages 1015–1020, 2008.

[11] D. Maslov, G. W. Dueck, and D. M. Miller. Techniques for the synthesis of reversible toffoli networks. *ACM Trans. Des. Autom. Electron. Syst.*, 12(4):42, 2007.

[12] M. Saeedi, M. Sedighi, and M. Saheb Zamani. A novel synthesis algorithm for reversible circuits. *IEEE/ACM International Conference on Computer-Aided Design*, pages 65–68, 2007.

[13] R. Wille and R. Drechsler. Effect of BDD optimization on synthesis of reversible and quantum logic. *Workshop on Reversible Computation*, 2009.

[14] P. Gupta, A. Agrawal, and N.K. Jha. An algorithm for synthesis of reversible logic circuits. *IEEE Trans. on CAD*, 25(11):2317–2330, 2006.

[15] Z. Sasanian, M. Saeedi, M. Saheb Zamani, and M. Sedighi. A cycle based synthesis algorithm for reversible logic. *Asia and South Pacific Design Automation Conference*, pages 745–750, 2009.

[16] A. K. Prasad, V. V. Shende, K. N. Patel, I. L. Markov, and J. P. Hayes. Data structures and algorithms for simplifying reversible circuits. *J. Emerg. Technol. Comput. Syst.*, 2(4), October 2006.

[17] D. Maslov, C. Young, D. M. Miller, and G. W. Dueck. Quantum circuit simplification using templates. *Design, Automation and Test in Europe*, pages 1208–1213, March 2005.

[18] D. Maslov, G. W. Dueck, and D. Michael Miller. Toffoli network synthesis with templates. *IEEE Trans. on CAD*, 24(6):807–817, 2005.

[19] M. Saeedi, M. Saheb Zamani, and M. Sedighi. On the behavior of substitution-based reversible circuit synthesis algorithms: Investigation and improvement. *International Symposium on VLSI*, pages 428–436, 2007.

[20] R. Wille, D. Große, L. Teuber, G. W. Dueck, and R. Drechsler. Revlib: An online resource for reversible functions and reversible circuits. *International Symposium on Multiple Valued Logic*, pages 220–225, May 2008.

[21] V. Kolmogorov. Blossom V: A new implementation of a minimum cost perfect matching algorithm. University College London, September 8, 2008.