Simplification of Toffoli Networks via Templates

Dmitri Maslov, Gerhard W. Dueck
Faculty of Computer Science
University of New Brunswick
Fredericton, N.B. E3B 5A3 CANADA

D. Michael Miller
Dept. of Computer Science
University of Victoria
Victoria, BC, V8W 3P6 Canada

Abstract

Reversible logic functions can be realized as networks of Toffoli gates. The synthesis of Toffoli networks can be divided into two steps. First, find a network that realizes the desired function. Second, transform the network such that it uses fewer gates, while realizing the same function. This paper addresses the second step. Transformations are accomplished via template matching. The basis for a template is a network with m gates that realizes the identity function. If a sequence in the network to be synthesized matches more than half of a template, then a transformation reducing the gate count can be applied. All templates for $m \leq 7$ are described in this paper.

1 Introduction

Reversible logic is an emerging research area. Interest in reversible logic is sparked by its applications in quantum computing, low-power CMOS, nanotechnology, and optical computing. The synthesis of reversible circuits differs significantly from synthesis using traditional irreversible gates. Two restrictions are added for reversible networks, namely fan-out and feedback are not allowed. The only possible structure for a reversible network is a cascade of reversible gates. The most frequently used gates are the Toffoli gate [15] and the Fredkin gate [4]. The Toffoli gate inverts a single bit if the AND of a set of control lines is 1. The Fredkin gate interchanges two bits if the AND of a set of control lines is 1. The formal definition of the Toffoli gate is given in Section 2.

Only a few synthesis methods have been proposed for reversible logic. Suggested methods include: using Toffoli gates to implement an ESOP (EXOR sum-of-products) [11], exhaustive enumeration [14, 13], heuristic methods that iteratively make the function simpler (simplicity is measured by the Hamming distance [2] or by spectral means [9]), and transformation based synthesis [5], among others. Some methods use excessive search time, others are not guaranteed to converge, and some require many additional outputs (garbage). We follow the two-step approach suggested in [10]. First a network for the given function is found. The algorithm for this step is guaranteed to converge. In fact, the algorithm is very fast. Improvements on a naive algorithm are described in [10]. The second step consists of applying transformations which reduce the number of gates. In this paper we describe the templates used for such transformations in detail.

Several authors considered network transformations. Shende *et. al* [14, 13] used several 4-bit circuit equivalences to be able to rewrite gates in a different order. Their circuit equivalence rules were not proposed for circuit simplification. In our work we cover and classify all templates they describe, generalize the notion of template, and show how to use them to simplify networks. Iwama, Kambayashi, and Yamashita [5] introduced some circuit transformation rules, which mainly served to bring a network to a canonical form and thus, stating that the set of transforms is complete. However, their approach uses a high number of garbage bits, whereas in our approach no garbage is allowed. One of the transforms in [5] was proposed for circuit simplification, but the actual application procedure was not described. Our work generalizes and classifies the templates used by [5], and adds new classes. We also show a way of using templates for network simplification, implemented it, and show results.

2 Preliminaries

An n-input n-output function (gate) is called **reversible** if, and only if, it maps each input instance to a unique output instance. In other words, a reversible function (gate) permutes the elements of its domain. In practice, not all of the $n!$ possible reversible functions can be realized as a single reversible gate. Several reversible gates have been proposed. However, we will only deal with Toffoli gates in this paper.

Definition 1. *For the set of domain variables* $\{x_1, x_2, \ldots, x_n\}$ *the* **generalized Toffoli gate** *has the form*

$TOF(C,t)$, *where* $C = \{x_{i_1}, x_{i_2}, ..., x_{i_k}\}, t = \{x_j\}$ *and* $C \cap t = \emptyset$, *and it maps the Boolean pattern* $\{x_1^0, x_2^0, ..., x_n^0\}$ *to* $\{x_1^0, x_2^0, ..., x_{j-1}^0, x_j^0 \oplus x_{i_1}x_{i_2}...x_{i_k}, x_{j+1}^0, ..., x_n^0\}$. *The set* C *which controls the change of* j*-th bit is called the set of* **control lines** *and* t *is called the* **target**.

In the literature, a subset of all generalized Toffoli gates is typically considered. The most popular are: the NOT gate ($TOF(\emptyset, x_j)$), a generalized Toffoli gate which has no controls; the CNOT gate ($TOF(x_i, x_j)$)[3], which is also known as a Feynman gate, a generalized Toffoli gate with one control bit, and the Toffoli gate $TOF(x_{i_1} + x_{i_2}, x_j)$ (where "+" denotes set union) [15], a generalized Toffoli gate with two controls. The three gates are illustrated in Figure 1, and the gates with more controls are drawn similarly. Note that the way the gates are drawn is a convention, which is not related to the way the gates are implemented. Gates with more than two controls are discussed in [6]. The set of generalized Toffoli gates is known to be complete (for example, see [8]), in other words, any reversible function can be realized as a cascade of Toffoli gates. A regular synthesis method for Toffoli gate networks is discussed in [10].

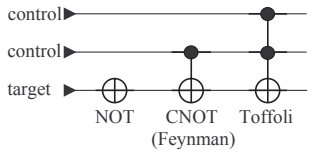

Figure 1. NOT, CNOT and Toffoli gates

Due to probable technological restrictions, the synthesis of reversible logic is done with no feedback and no fan-out [12]. This leaves the cascade structure as the only model satisfying those conditions. Thus, we consider cascades of Toffoli gates.

Let the signal be propagated from left to right. The pictorial representation of a network is shown in Figure 2. The **cost** of a function is defined as the number of gates in the circuit realizing it (S for a network in Figure 2).

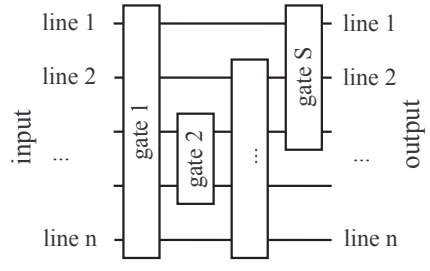

Figure 2. The general structure for a network

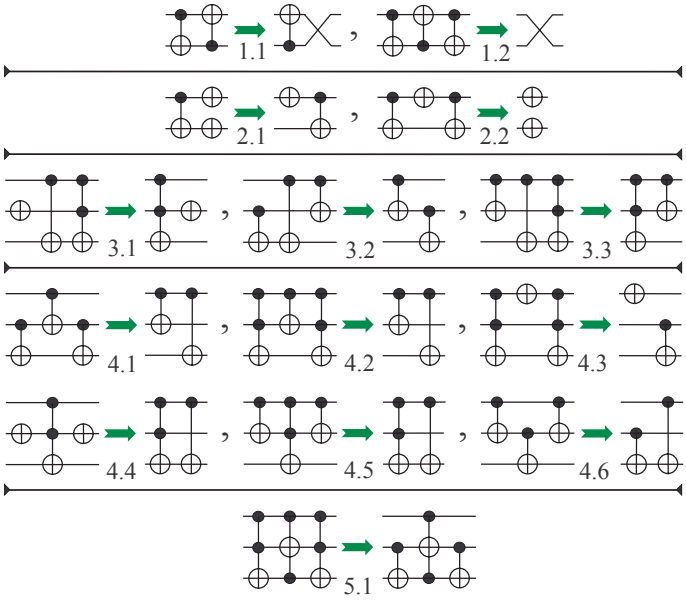

Figure 3. Templates with 2 or 3 inputs.

3 Templates

Previously [10], templates were introduced as a tool for network simplification. In that work, a template consists of two sequences of gates which realize the same function. The first sequence of gates is to be matched to a part of the circuit being simplified and the second sequence is to be substituted when a match is found. The templates were in Figure 3 were identified and classified based on their similarity.

In [10], the template matching procedure looks for the first set of gates, including the initial match to the widest gate, across the entire circuit. If all target gates are found, it attempts to make them adjacent using the **moving rule**: gate $TOF(C_1, t_1)$ can be interchanged with gate $TOF(C_2, t_2)$ if, and only if, $C_1 \cap t_2 = \emptyset$ and $C_2 \cap t_1 = \emptyset$. Adjacent gates can match the template in the forward or reverse direction. The matched gates are replaced with the new gates specified by the template. For a reverse match, the new gates are substituted in reverse order. Finally, if at any time two adjacent gates are equal, they can be deleted, (**deletion rule**).

In this section, we give a formal classification of the templates used in [10]. For a better understanding of template classes, we introduce the following notation.

- the left hand side has a sequence of gates that is to be replaced with the sequence given on the right hand side;
- the controls of the gates are coded by sets C_i each of which represents a set (maybe empty) of lines;

- the target sets t_i each contain a single line.

  All sets are disjoint: $C_i \cap C_j = \emptyset, C_i \cap t_k = \emptyset, t_l \cap t_k = \emptyset \ \forall i, j, k, l$.

Using this notation, a **class** of templates can be defined as a set of templates which can be described by one formula. A first attempt to classify the templates results in the classes listed below:

Class 1. This class unites and generalizes the templates 2.2, 4.1-4.3 (Figure 3) into a class (Figure 4a) with the formula:

$$\begin{aligned} &TOF(C_1 + C_2 + t_2, t_1)\ TOF(C_1 + C_3, t_2) \\ &\qquad TOF(C_1 + C_2 + t_2, t_1) = \\ &= TOF(C_1 + C_3, t_2)\ TOF(C_1 + C_2 + C_3, t_1) \end{aligned} \quad (1)$$

Class 2. This class consists of templates 4.4-4.6 (Figure 3) and their generalizations. The class is illustrated in Figure 4b and can be written as the following formula:

$$\begin{aligned} &TOF(C_1 + C_2, t_2)\ TOF(C_1 + C_3 + t_2, t_1) \\ &\qquad TOF(C_1 + C_2, t_2) = \\ &= TOF(C_1 + C_3 + t_2, t_1)\ TOF(C_1 + C_2 + C_3, t_1) \end{aligned} \quad (2)$$

Class 3. This class (Figure 4c) includes templates 2.1, 3.1-3.3 (Figure 3) and can be described by the formula:

$$\begin{aligned} &TOF(C_1 + C_2 + t_2, t_1)\ TOF(C_1 + C_2 + C_3, t_1) \\ &\qquad TOF(C_1 + C_3, t_2) = \\ &= TOF(C_1 + C_3, t_2)\ TOF(C_1 + C_2 + t_2, t_1) \end{aligned} \quad (3)$$

Template 5.1 can be generalized, but this generalization is not considered here since template 5.1 does not decrease the number of gates in a network. However, use of a generalization of this template may be beneficial since it introduces smaller gates that can be used by other templates. Even if they are not used, it is beneficial to have gates with fewer controls, since for some technologies their costs are lower. For instance, in quantum technology the cost of a Toffoli gate is 7 times higher than that of a CNOT gate [1]. As the number of controls of the Toffoli gate grows, the relation between the costs of generalized Toffoli and CNOT gate grows quadratically if no additional garbage is allowed and linearly if garbage is allowed [1].

The correctness of formulas (1)-(3) is easily proven. A more interesting question is whether the set of these three classes of templates together with the two rules (moving rule, deletion rule) is a complete set of simplification rules for a sequence of three generalized Toffoli gates over n lines. To check this, we ran a program which exhaustively searches all sequences of three gates built on three lines to check whether the sequence can be reduced by means of templates from the three classes and the two rules. This program found no new templates. Thus, we conclude that the three classes together with moving and deletion rules form the complete simplification tool for any Toffoli network with up to three gates.

4 Templates - a New Approach

Further we generalize the template tool, but assume the following limitations: the model gates should necessarily be self-inverses. This limits the generality of a template, but for the goals of the current paper this does not change the essence, since a generalized Toffoli gate is a self-inverse.

Although the template description in Section 3 is formal and shorter (3 classes and 2 rules in comparison to 14 templates with 2 rules as used before), it can be simplified even further. For this we need a new understanding of templates.

Let a **size** m **template** be a sequence of m gates (a circuit) which realizes the identity function. Any template of size m must be independent of templates of smaller size, *i.e.* for a given template size m no application of any set of templates of smaller size can decrease the number gates. The template $G_0\ G_1 \ldots G_{m-1}$ can be applied in two directions:

1. **Forward application:** A piece of network that matches the sequence of gates $G_i\ G_{(i+1) \bmod m} \cdots\ G_{(i+k-1) \bmod m}$ of the template $G_0\ G_1 \ldots G_{m-1}$ exactly, is replaced with the sequence $G_{(i-1) \bmod m}\ G_{(i-2) \bmod m} \cdots\ G_{(i+k) \bmod m}$ without changing the network's output, where $k \in N, k \geq \frac{m}{2}$.

2. **Backward application:** A piece of network that matches the sequence of gates $G_i\ G_{(i-1) \bmod m} \cdots\ G_{(i-k+1) \bmod m}$ exactly, is replaced with the sequence $G_{(i+1) \bmod m}$ $G_{(i+2) \bmod m} \ldots G_{(i-k) \bmod m}$ without changing the network output, where $k \in N, k \geq \frac{m}{2}$.

These definitions of template application need a correctness proof—the network output should not be changed for each of the listed operations. Correctness can be verified as follows. Note, that a reversible cascade that realizes a function f read in reverse (from the outputs to the inputs) realizes f^{-1}, its inverse.

First, we prove the correctness of the forward application of a template starting with element G_0. The operation for this case requires substitution of $G_0\ G_1 \ldots G_{k-1}$ with $G_{m-1}\ G_{m-2} \ldots G_k$. Since $G_0\ G_1 \ldots G_{m-1}$ realizes the identity function, $G_k\ G_{k+1} \ldots G_{m-1}$ realizes the inverse of the function realized by $G_0\ G_1 \ldots G_{k-1}$. Therefore, read in reverse order $G_k\ G_{k+1} \ldots G_{m-1}$ realizes the inverse of the inverse, *i.e.* the function itself. Thus, the function realized by $G_0\ G_1 \ldots G_{k-1}$ was substituted by itself, which does not change the output of the network. Correctness of

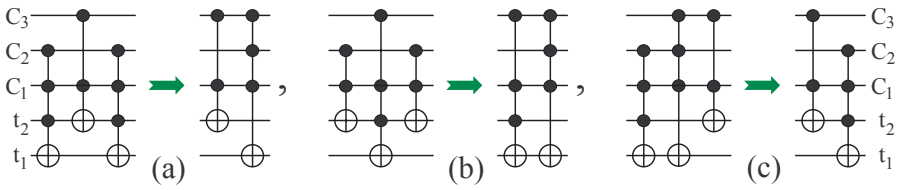

Figure 4. Toffoli templates.

the remaining forward applications can be proven by using Lemma 1.

Correctness of all reverse applications follows from the proof above and from the observation that the inverse of the identity function is the identity function.

Next, observe that a template can be used in both directions, forward and backward as the formulas show. Also, we can start using it from any element. Thus, it is better to think of a template as a cyclic sequence. The correctness of viewing a template as a cyclic sequence is proven with the following Lemma.

Lemma 1. *If a network $G_0\ G_1 \ldots G_{m-1}$ realizes the identity function, then for any k-shift, $G_k\ G_{(k+1) \bmod m} \cdots G_{(k-1) \bmod m}$ realizes the identity.*

Proof. We prove the Lemma for 1-shift, $G_1\ G_2 \ldots G_{m-1}\ G_0$. Then all k-shifts can be proven by applying the 1-shift k times. The proof for a 1-shift follows from:

$$\begin{aligned} Id &= G_0\ G_1 \ldots G_{m-1} \\ G_0\ Id &= G_0\ G_0\ G_1 \ldots G_{m-1} \\ G_0 &= G_1\ G_2 \ldots G_{m-1} \\ Id = G_0\ G_0 &= G_1\ G_2 \ldots G_{m-1}\ G_0. \end{aligned}$$

□

The condition $k \geq \frac{m}{2}$ is used as we do not want to increase the number of gates when a template is applied and equality yields a simpler classification scheme.

The following is a classification of templates up to size 7. We use the notation introduced in the previous section.

- m=1. Size 1 templates do not exist, since each generalized Toffoli gate produces a change of its input.

- m=2. There is one class of templates of size 2 (Figure 5a), and it is the deletion rule which is described by the sequence (AA)

$$TOF(C_1, t_1)\ TOF(C_1, t_1).$$

- m=3. There are no templates of size 3.

- m=4. There is one class of templates (Figure 5b), the moving rule from the previous section, which can be written as follows (ABAB):

$$\begin{aligned} &TOF(C_1 + C_2, C_4 + C_5)\ TOF(C_1 + C_3, C_4 + C_6) \\ &TOF(C_1 + C_2, C_4 + C_5)\ TOF(C_1 + C_3, C_4 + C_6). \end{aligned}$$

  The set notation is used to describe the targets since they may intersect or not, which is impossible to describe in one formula using the t_i notation for the targets. The upper template in Figure 5b has $|C_4| = 0$ which results in $|C_5| = 1$ and $|C_6| = 1$, when the lower has $|C_4| = 1$ resulting in $|C_5| = 0$ and $|C_6| = 0$.

- m=5. Surprisingly, there is only class of template of size 5 (Figure 5c), which unites the three earlier classes (1)-(3) and includes templates 2.1-2.2, 3.1- 3.3 and 4.1-4.6 from Figure 3. The class can be written as (ABABC):

$$\begin{aligned} &TOF(C_1 + C_2 + t_2, t_1)\ TOF(C_1 + C_3, t_2) \\ &TOF(C_1 + C_2 + t_2, t_1)\ TOF(C_1 + C_3, t_2) \\ &TOF(C_1 + C_2 + C_3, t_1). \end{aligned}$$

- m=6. There are two classes here (Figure 5d), and they are described by formulas (ABACBC)

$$\begin{aligned} &TOF(C_1 + t_2, t_1)\ TOF(C_1 + C_2 + C_3 + t_1, t_2) \\ &TOF(C_1 + t_2, t_1)\ TOF(C_1 + C_2 + t_2, t_1) \\ &TOF(C_1 + C_2 + C_3 + t_1, t_2)\ TOF(C_1 + C_2 + t_2, t_1) \end{aligned}$$

  and (ABACDC)

$$\begin{aligned} &TOF(C_1 + t_2, t_1)\ TOF(C_1 + C_2 + C_3 + t_1, t_2) \\ &TOF(C_1 + t_2, t_1)\ TOF(C_1 + C_2 + t_1, t_2) \\ &TOF(C_1 + C_2 + C_3 + t_2, t_1)\ TOF(C_1 + C_2 + t_1, t_2). \end{aligned}$$

  Note, the two formulas for the classes look very similar, and, in fact using Fredkin gates, they can be generalized to form one very simple template $FRE(C_1 + C_2 + C_3, t_1 + t_2)\ FRE(C_1 + C_2 + C_3, t_1 + t_2)$ (where $FRE(C, t_1 + t_2)$ is a gate which swaps values of bits t_1 and t_2 if, and only if, set C has all ones on its lines), but we do not pursue this here as we are restricting our attention to generalized Toffoli gates.

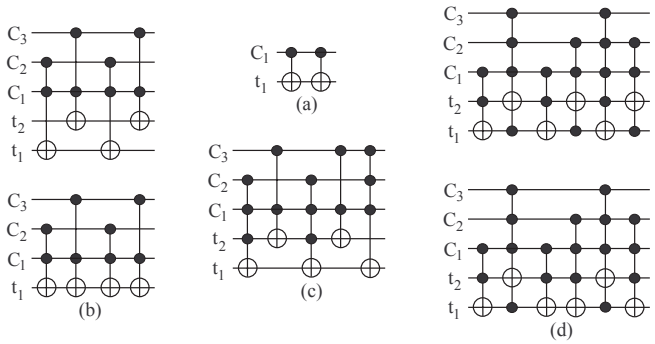

Figure 5. All templates for $m \leq 7$.

- m=7. There are no templates of size 7.

To verify the correctness of the above classification, we must show that no template of larger size can be reduced to a template of smaller size, which can be done by hands.

4.1 Completeness

First, we wrote a program which builds all the 4-input 4-output circuits of size 7 that realize the identity function and tries to apply the templates. The program result shows that the set of our 5 templates (AA, ABAB, ABABC, ABACBC, ABACDC) is the complete set of templates of size 7 or less for 4 inputs and less.

The mathematical proof of completeness of this set for any number of inputs is harder. For templates of size 2 it can be done by hand, since there are not so many choices to look at. For templates of size 4 and 5 the following lemmas (proven in [7]) are useful.

Lemma 2. *A size m template has at most $\lfloor \frac{m}{2} \rfloor$ different lines with EXOR signs.*

Lemma 3. *Given a template of size m where all gates have the same target line (i.e. the EXOR appears on a single line), then m is even and all gates can be grouped as pairs of equal gates.*

Use of Lemma 2 allows us to say that all the templates of size 4 have EXOR signs on either two lines (two signs on one and two on the other) or 1 line (all 4 on 1 line). In the last case use Lemma 3. Thus, an exhaustive search proof becomes reasonable. For the size 5 templates we can guarantee that they all will have exactly two lines with EXOR. Note, that Lemma 3 proves that the only two templates which have only one line affected with EXOR are the duplication deletion rule and the passing rule, all other identities of this type are applications of the above rules.

Size	(a)	(b)	(c)	(d)
17				
16				
15	2			
14	22	8		
13	112	43	10	
12	432	215	91	
11	1191	651	410	
10	2575	1776	1458	
9	5116	4038	3846	
8	7842	7405	7412	577
7	8989	9716	10082	10253
6	7478	8573	8977	17049
5	4314	5167	5294	8921
4	1682	2055	2066	2780
3	463	558	559	625
2	89	102	102	102
1	12	12	12	12
0	1	1	1	1
WA:	7.25	6.92	6.83	5.87

(a): before template matching
(b): after template matching
(c): modified template matching
(d): optimal sizes [14, 13]

Table 1. Number of reversible functions using a specified number of gates for $n = 3$.

5 Experimental Results

We wrote programs to verify the correctness of our results, build the new templates and apply them. The results of the verification program were discussed in above.

The program which simplifies the networks works as follows. First, it is convenient to store template ABAB as a separate rule which helps to bring the gates together to match a template. Then, the circuit is simplified as follows. We call this the "moving rule". For the hierarchy of templates AA $\succ$ ABABC $\succ$ ABACBC $\succ$ ABACDC try to match as many gates of a template as possible by looking ahead in the network and using the moving rule. If a template can be applied, apply it for the greatest application parameter k possible. After applying any template start trying to apply the templates in hierarchical order from the very beginning. If none of the templates can be applied, the simplification process stops.

The template simplification tool was tested on all size 3 reversible functions synthesized by the algorithm from [10]. We also investigated a *modified template matching* procedure. It works as follows. Normally, a section of the circuit is only replaced when more than half of the gates

in the templates are matched. However, it may be advantageous to replace the gates if exactly half of the gates match, provided that the gates that replace them have fewer control lines. Empirical results show that this can lead to reductions in the size of the circuit. Further reduction of the size of synthesized networks can be achieved when template simplification tool is applied in conjunction with simulated annealing technique. However, the results reported in Table 1 are good themselves: more than 50% of the synthesized networks are optimal.

Table 1 shows the number of functions for each gate count and the average number of gates required.

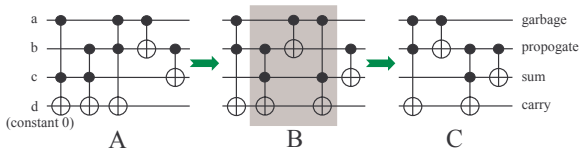

Figure 6. Optimal circuit for a full adder.

Example 1. *We took a network for three bit adder produced by the synthesis algorithm presented in [10] (Figure 6A) and applied our program to simplify it. As expected, the program used a size 5 template and matched 3 gates (Figure 6B). Thus, they were substituted by the remaining 2 gates of the template read in reverse order (Figure 6C). This circuit is optimal, since no further reduction is possible. Suppose, an adder can be realized with 3 gates or less. Then, addition of these gates to the end of the built size 4 cascade results in a new template which was proven (by enumeration) not to exist for size 7 and less and four inputs.*

6 Conclusion

The larger the set of templates, the more reductions can be done. For instance, if for some natural number k k-optimality is defined as the impossibility of simplifying a network with size $2k-1$ and less templates, then all the templates of size $n * 2^{n+1} - 1$ and less form the complete simplification tool for the synthesis method provided in [10]. The theoretical algorithm from [10] produces a valid network with at most $n * 2^n$ gates, therefore if this network is not optimal and was not simplified by all templates with size $n * 2^{n+1} - 1$ and less, not all the templates are listed. Thus, we come to a contradiction which proves the statement.

In this work, we built the set of templates and showed a procedure allowing us to create 4-optimal circuits for networks with number of inputs less than or equal to 4. We generalized these templates and proposed them as the set of rules which produces a 4-optimal network out of those given. The template tool was generalized and shown in a readily usable form.

References

[1] A. Barenco, C. H. Bennett, R. Cleve, D. P. DiVinchenzo, N. Margolus, P. Shor, T. Sleator, J. A. Smolin, and H. Weinfurter. Elementary gates for quantum computation. *The American Physical Society*, 52:3457–3467, 1995.

[2] G. W. Dueck and D. Maslov. Reversible function synthesis with minimum garbage outputs. In *6th International Symposium on Representations and Methodology of Future Computing Technologies*, pages 154–161, March 2003.

[3] R. Feynman. Quantum mechanical computers. *Optic News*, 11:11–20, 1985.

[4] E. Fredkin and T. Toffoli. Conservative logic. *International Journal of Theoretical Physics*, 21:219–253, 1982.

[5] K. Iwama, Y. Kambayashi, and S. Yamashita. Transformation rules for designing CNOT-based quantum circuits. In *Design Automation Conference*, New Orleans, Louisiana, USA, June 10-14 2002.

[6] D. Maslov and G. Dueck. Asymptotically optimal regular synthesis of reversible networks. In *International Workshop on Logic Synthesis*, pages 226–231, Laguna Beach, CA, 2003.

[7] D. Maslov, G. Dueck, and M. Miller. Templates for Toffoli network synthesis. In *International Workshop on Logic Synthesis*, pages 320–326, Laguna Beach, CA, 2003.

[8] D. Maslov and G. W. Dueck. Garbage in reversible design of multiple output functions. In *6th International Symposium on Representations and Methodology of Future Computing Technologies*, pages 162–170, March 2003.

[9] D. M. Miller and G. W. Dueck. Spectral techniques for reversible logic synthesis. In *6th International Symposium on Representations and Methodology of Future Computing Technologies*, March 2003.

[10] D. M. Miller, D. Maslov, and G. W. Dueck. A transformation based algorithm for reversible logic synthesis. In *Proceedings of the Design Automation Conference*, pages 318–323, June 2003.

[11] A. Mishchenko and M. Perkowski. Logic synthesis of reversible wave cascades. In *International Workshop on Logic Synthesis*, June 2002.

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

[13] V. Shende, A. Prasad, I. Markov, and J. Hayes. Synthesis of reversible logic circuits. *IEEE Trans. CAD*, 22(6):723–729, June 2003.

[14] V. V. Shende, A. K. Prasad, I. L. Markov, and J. P. Hayes. Reversible logic circuit synthesis. In *International Conference on Computer Aided Design*, pages 125–132, San Jose, California, USA, Nov 10-14 2002.

[15] T. Toffoli. Reversible computing. *Tech memo MIT/LCS/TM-151, MIT Lab for Comp. Sci*, 1980.