Toffoli Network Synthesis with Templates

Dmitri Maslov, Gerhard W. Dueck, *Member, IEEE*, and D. Michael Miller, *Member, IEEE*

***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 above synthesis approach.**

We present a basic method and, based on that, a bidirectional synthesis algorithm which produces a network of Toffoli gates realizing a given reversible specification. An asymptotically optimal modification of the basic synthesis algorithm employing generalized mEXOR gates is also presented.

Transformations are then applied using template matching. The basis for a template is a network of gates that realizes the identity function. If a sequence of gates in the synthesized network matches a sequence comprised of more than half the gates in a template, then a transformation using the remaining gates in the template can be applied resulting in a reduction in the gate count for the synthesized network. All templates with up to 6 gates are described in this paper.

Experimental results including an exhaustive examination of all 3 variable reversible functions and a collection of benchmark problems are presented. The paper concludes with suggestions for further research.

***Index Terms*— Logic synthesis, reversible logic, quantum computing.**

I. Introduction

INTEREST in reversible logic is motivated by its applications in quantum computing, low-power CMOS, nanotechnology, and optical computing. Synthesis of reversible networks is an emerging research topic. It differs significantly from synthesis using traditional irreversible gates. Two restrictions are added for reversible networks, namely fan-out and feed-back are not allowed. Thus, the only possible structure for a reversible network is a cascade of reversible gates. Toffoli gates [17] are the most frequently used and best investigated. The Toffoli gate inverts a single bit if the AND of a set of control lines is 1. The formal definition is given in Section II.

Only a few synthesis methods have been proposed for reversible logic including: using Toffoli gates to implement an ESOP (EXOR sum-of-products) [13], exhaustive enumeration [15], heuristic methods that iteratively make the function simpler (simplicity is measured by the Hamming distance [2] or by spectral means [11]), and transformation based synthesis [5], [16], among others. Some methods use excessive search time, others are not guaranteed to converge, and some require many additional *garbage* outputs.

Manuscript received ???, 2004; revised ???, 2004. The work of G. Dueck was supported in part by a grant form the Natural Sciences and Engineering Research Council of Canada.

D. Maslov and M. Miller are with the Department of Computer Science, University of Victoria, Victoria, BC, Canada, V8W 3P6 (dmitri.maslov@unb.ca, mmiller@cs.uvic.ca).

G. Dueck is with the Faculty of Computer Science, University of New Brunswick, Fredericton, NB, Canada, E3B 5A3 (gdueck@unb.ca).

We follow the two-step approach suggested in [12]. First a network that uses minimal garbage is found for a given function. The algorithm for this step is guaranteed to converge. It uses no backtracking or look-ahead and is thus very fast. The second step consists of applying transformations which reduce the number of gates in the network synthesized in the first step.

Several authors have considered transformations for reversible networks. Shende *et. al* [15], [16] used several 4-bit network equivalences to be able to rewrite a limited set of gates in a different order. In our work we cover and classify all rewriting rules they describe, use more gates for our template construction, generalize the notion of rewriting rules with templates, and show how to use templates to simplify networks. Iwama, Kambayashi, and Yamashita [5] introduced some Toffoli network transformation rules, which mainly served to bring a network to a canonical form. They stated that their set of transforms is complete. However, their approach uses a high number of garbage bits, whereas in our approach no extra garbage outputs (other than those required to achieve a reversible specification [10]) are used. The transforms in [5] were proposed for network simplification, but the actual application procedure was not described. Our work generalizes and classifies the rules used in [5], and adds new classes. In this paper, we show how templates can be applied to network simplification and show results of doing that.

Section 2 presents the background on reversible logic and Toffoli gates necessary for this paper. Section 3 presents a basic synthesis algorithm and an extension to bidirectional synthesis. An asymptotically optimal modification to the basic algorithm which uses mEXOR gates is also given. Templates and our approach to template matching are describe in section 4. Experimental results are presented in sections 5 and 6 and the paper concludes with some observations and suggestions for further research in section 7.

II. Preliminaries

An n-input n-output function (gate) is termed **reversible** if it is a bijection. In other words, a reversible function (gate) permutes the elements of its domain. In practice, not all of the $2^n!$ possible reversible functions can be realized as a single reversible gate. Several reversible gates have been proposed. Toffoli gates [17] and Fredkin gates [4] are the best known and most widely studied. We will only consider Toffoli gates in this paper.

Definition 1: For the set of domain variables $\{x_1, x_2, ..., x_n\}$ 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 $C \cap t = \emptyset$. It maps the Boolean pattern $\{x_1^0, x_2^0, ..., x_n^0\}$ to $\{x_1^0, x_2^0, ..., x_{j-1}^0, x_j^0 \oplus x_{i_1}^0 x_{i_2}^0 ... x_{i_k}^0, x_{j+1}^0, ..., x_n^0\}$. The set

C which controls the change of the 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 with 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) [17], a generalized Toffoli gate with two controls. The three gates are illustrated in Figure 1. 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. The set of generalized Toffoli gates is known to be complete (for example, see [10]), in other words, any reversible function can be realized as a cascade of generalized Toffoli gates and no additional garbage outputs are required. For simplicity, we will use Toffoli gate to mean *generalized* Toffoli gate throughout this paper.

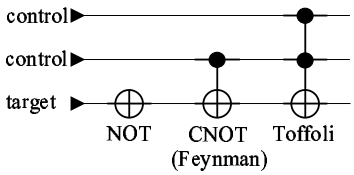

Fig. 1. NOT, CNOT and Toffoli gates

Synthesis of reversible logic is often considered in conjunction with quantum technologies [14]. Due to quantum mechanical restrictions, the synthesis of reversible logic is done with no feed-back and no fan-out [14]. Fan-outs and feed-backs are also not allowed for some other technologies that use reversible gates. This leaves the cascade structure as the only network topology 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 **reversible cost** (or simply, **cost**) of a function implementation is defined as the number of gates in the network realizing it (S for the network in Figure 2).

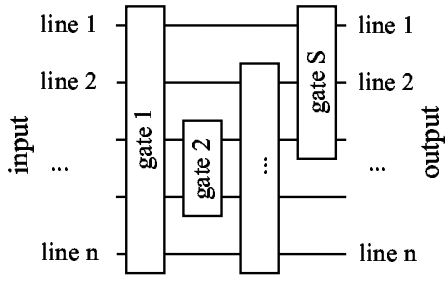

Fig. 2. The general structure for a reversible network

III. Synthesis Algorithm

To begin, we present a basic greedy algorithm. We consider a reversible function specified as a mapping over $\{0, 1, ..., 2^n - 1\}$; in other words, a truth vector. We write a function as $f(i)$, where i is an integer in the range $0 \leq i < 2^n - 1$, meaning that the function argument i is a vector giving the binary expansion of the integer i. The result of the function application to an integer argument i, $f(i)$, is treated as an integer as well. The basic algorithm works by assigning Toffoli gates at the output end of the cascade. The Toffoli gates are chosen so that the output part of the specification is progressively transformed to match the input part. When a cascade of Toffoli gates that transforms the total specification to the identity is found, then read in reverse order this cascade transforms the input to the required output, that is, it realizes the target function.

Basic Algorithm.

Step 0: If $f(0) \neq 0$, invert the outputs corresponding to 1-bits in $f(0)$. Each inversion requires a NOT gate. The transformed function, written as f^+, has $f^+(0) = 0$.

Step i: Consider each i in turn for $1 \leq i < 2^n - 1$ letting f^+ denote the current reversible specification. If $f^+(i) = i$, no transformation and hence no Toffoli gate is required for this i. Otherwise, gates are required to transform the specification to a new specification f^{++} with $f^{++}(i) = i$. The required gates must map $f^+(i) \rightarrow i$.

Let p be the bit string with 1s in all positions where the binary expansion of i is 1 while the expansion of $f^+(i)$ is 0. These are the 1 bits that must be added in transforming $f^+(i) \rightarrow i$. Conversely, let q be the bit string with 1s in all positions where the expansion of i is 0 while the expansion of $f^+(i)$ is 1. q identifies the 1 bits to be removed in the transformation.

For each $p_j = 1$, apply the Toffoli gate with control lines corresponding to all outputs in positions where the expansion of i is 1 and whose target line is the output in position j. This will increase the lexicographical order of $f^+(i)$. Then, for each $q_k = 1$, apply the Toffoli gate whose target line is the output in position k, and with control lines corresponding to all outputs in positions, except k, where the expansion of $f^+(i)$ is 1. This second operation decreases the lexicographical order but not below i.

Correctness analysis. For each $1 \leq i < 2^n - 1$, Step i transforms $f^+(i) \rightarrow i$ by applying the specified sequence of Toffoli gates. Since we consider the i values in increasing order, and Step 0 handles the case for $f(0)$, we know that $f^+(j) = j, 0 \leq j < i$. The importance of this is that it shows that none of the Toffoli gates generated in a decreasing order step affect $f^+(j), j < i$. In other words, once a row of the specification is transformed to the correct value, it will remain at that value regardless of the transforms required for later rows. Clearly, the final row of the specification never requires a transformation as it is correct by virtue of the correct placement of the preceding $2^n - 1$ values.

Table I illustrates the application of the basic algorithm. (i) is the given specification. Step 0 identifies the application of $TOF(a^0)$ giving (ii). At this point $f^+(i), 0 \leq i \leq 4$ are as required. Mapping $f^+(5) \rightarrow 5$ requires $TOF(c^1 + b^1, a^1)$ to change the rightmost bit to 1 (iii) and $TOF(c^2 + a^2, b^2)$ to remove the center 1 (iv). Lastly, $TOF(c^3+, b^3, a^3)$ is again required, this time to map $f^+(6) \rightarrow 6$. Note that the gates are identified in order from the output side to the input side. The

TABLE I
EXAMPLE OF APPLYING THE BASIC ALGORITHM

	(i)	(ii)	(iii)	(iv)	(v)
cba	$c^0b^0a^0$	$c^1b^1a^1$	$c^2b^2a^2$	$c^3b^3a^3$	$c^4b^4a^4$
000	001	00**0**	000	000	000
001	000	00**1**	001	001	001
010	011	01**0**	010	010	010
011	010	01**1**	011	011	011
100	101	10**0**	100	100	100
101	111	11**0**	11**1**	10**1**	101
110	100	10**1**	101	11**1**	11**0**
111	110	11**1**	11**0**	110	11**1**

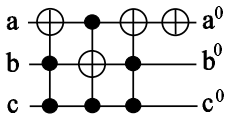

Fig. 3. Network for the function shown in Table I

corresponding network is shown in Figure 3.

The basic algorithm is straightforward and easily implemented. It is also easily seen that it will always terminate successfully with a network for the given specification.

Theorem 1: The basic algorithm successfully terminates giving a network of size less than or equal to $(n-1)2^n+1$ gates.

Proof: Note that each gate application brings at least one bit to its correct place, therefore the algorithm will definitely terminate after applying a maximum of $n2^n$ gates. In order to prove a tight upper bound, we construct a worst case function for this algorithm.

The first output pattern will require the maximum number of gates (n) to bring it to the form of the first input pattern, 0, if it is (2^n-1), the bitwise negation of 0. After applying the n NOT gates for the first pattern, assume the second output pattern is (2^n-2), the bitwise negation of the second input pattern 1. Again, n gates are needed . In the same manner, keep assuming that after applying the gates from previous steps, the output pattern for the current step is the negation of the input for that step, which can be done until step $(2^{n-1}-1)$ of the algorithm is completed.

At this point, the first 2^{n-1} input patterns match the input, so the most significant bit of the output patterns has been completely dealt with (2^{n-1} zeros are in the upper part of the truth table, the lower 2^{n-1} must then by definition be 1). Therefore, starting from this step, the most significant bit is fixed, and we cannot negate it to create a desired difficult pattern. Starting from step (2^{n-1}), negate only the remaining $(n-1)$ unspecified bits of the output.

Similarly, at step $2^{n-1}+2^{n-2}$ of the algorithm the second most significant bit will be completely specified. In general, at step $2^{n-1}+2^{n-2}+...+2^{n-k}$, the k most significant bits are completely specified. Thus, the maximum number of gates produced by the algorithm becomes

$$\begin{aligned}
&n2^{n-1}+(n-1)2^{n-2}+(n-2)2^{n-3}+... \\
&\quad ...+2*2^{n-n+1}+1*2^{n-n} \\
&= nx^{n-1}+(n-1)x^{n-2}+(n-2)x^{n-3}+... \\
&\quad ...+2*x^1+1*x^0|_{x=2} \\
&= (x^n+x^{n-1}+x^{n-2}+...+x^2+x+1)'|_{x=2} \\
&= \Big(\frac{x^{n+1}-1}{x-1}\Big)'\Big|_{x=2} \\
&= \Big(\frac{(n+1)x^n(x-1)-x^{n+1}+1}{(x-1)^2}\Big)\Big|_{x=2} \\
&= \Big(\frac{(n+1)2^n(2-1)-2^{n+1}+1}{(2-1)^2}\Big) \\
&= (n+1)2^n-2^{n+1}+1=(n-1)2^n+1.
\end{aligned}$$

The above construction shows this bound is achievable. ■

Using the procedure from Theorem 1, it is possible to construct a (unique) function for any n that requires $(n-1)2^n+1$ gates. Therefore, the upper bound is tight. For $n=3$, this function has the specification (written as the vector of outputs for the input set $\{0,1,...,2^n-1\}$) $[7,1,4,3,0,2,6,5]$ and will be referred to as 3_17 since using Theorem 1, it can be calculated that its cost is $(3-1)2^3+1=2*8+1=17$. Analogously, 4_49 has the highest cost for 4 input functions. Its specification is $[15,1,12,3,5,6,8,7,0,10,13,9,2,4,14,11]$. Functions with a larger number of variables can be built. We will simplify the networks further and consider these worst-case functions to measure the level of improvement.

We next consider modifications to the basic algorithm, which usually result in smaller networks in terms of the number of gates and control inputs to particular gates.

A. Control Input Reduction

The basic algorithm naively assigns the maximum number of control lines to each Toffoli gate. Often a subset of those control lines will suffice. The requirement is that the gate does not affect rows earlier in the specification, *i.e.* rows that are already in the correct place. This is easily accounted for since the set of control lines must either contain a line that has not appeared as a 1 in any earlier row of the specification, or must contain all lines that have appeared as 1's in rows earlier in the specification. Given that, the revised algorithm, instead of using the control lines identified by the basic algorithm, considers all valid subsets of those lines, and chooses the control that minimizes the complexity $C(f^+)$ of the resulting specification. We take C to be the total Hamming distance between the input and output sides of the specification, so this heuristic chooses the gate that moves the specification furthest towards (lesser value of the Hamming distance) the identity specification. In the case of a tie, the smallest set of control lines is used, and within that the choice is arbitrary.

B. Bidirectional Algorithm

As described thus far, the algorithm produces the network by selecting Toffoli gates that manipulate the output side of the specification. Since the specification is reversible, one

TABLE II
EXAMPLE OF APPLYING THE BIDIRECTIONAL ALGORITHM

	(i)	(ii)	(iii)	(iv)
cba	$c^0b^0a^0$	$c^1b^1a^1$	$c^2b^2a^2$	$c^3b^3a^3$
000	111	000	000	000
001	000	111	001	001
010	001	010	010	010
011	010	001	111	011
100	011	100	100	100
101	100	011	101	101
110	101	110	110	110
111	110	101	011	111

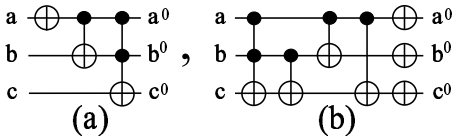

Fig. 4. Networks for the function shown in Table II

could consider the inverse specification, derive a reverse network, and then choose whichever network is smaller. A better approach is to apply the method in both directions simultaneously choosing to add gates at either the input side or the output side during each synthesis step.

To see how this works, consider the initial reversible specification in Table II, column (i). The basic algorithm would require that we invert each of a^0, b^0 and c^0 to make $f^+(0) = 0$. The alternative is to invert a, *i.e.* to apply the gate $TOF(a)$ to the input side. Applying this gate, and then reordering the specification so that the input side is again in standard truth-table order yields the specification in (ii). From the output side, we would next have to map $f^+(1) = 7 \rightarrow 1$. However, from the input side we can accomplish what is required by interchanging rows 1 and 3, which is done by applying the gate $TOF(a, b)$. Doing so, and reordering the input side into standard order, yields the specification in (iii). At this point, selection from the output side and the input side identify the same gate $TOF(a+b, c)$ (when expressed in terms of the input lines) and the network is done (iv). The result uses three gates (shown in Figure 4(a)), whereas approaching the problem from the output side alone requires three NOT gates just to handle $f(0)$ and seven gates in total (shown in Figure 4(b)).

In general, when $f^+(i) \neq i$, the choice is (a) to apply Toffoli gates to the outputs to map $f^+(i) \rightarrow i$, or (b) to apply Toffoli gates to the inputs to map $j \rightarrow i$ where j is such that $f^+(j) = i$. Since we consider the i in order, there must always be a j such that $j > i$. Also, the same rules for identifying the control lines, including the reduction described above apply. Let the bidirectional algorithm choose (a) if $H(i, f^+(i)) \leq H(i, j)$, and (b) otherwise (where $H(p, q)$ is the Hamming distance of the bit strings p and q). We thus base the choice on the number of gates required and not their width or how closely they map the specification to the identity.

C. Asymptotically Optimal Modification

In this subsection we provide an asymptotically optimal modification of the basic synthesis algorithm. For this modification, we need a new set of gates which are a further geralization of the basic Toffoli gate.

Definition 2: An **mEXOR gate** $TOF(C, T)$, where $C \cap T = \emptyset$ and $T = \{x_{j_1}, x_{j_1}, ..., x_{j_m}\}$ is a single gate that is equivalent to the network $TOF(C, x_{j_1})\ TOF(C, x_{j_2})...\ TOF(C, x_{j_m})$.

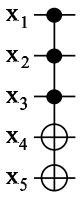

Fig. 5. Example of an mEXOR Toffoli gate

A pictorial representation of a mEXOR gate is shown in Figure 5.

Given the set of mEXOR gates, the synthesis procedure works as follows. The idea is to use the Toffoli gate algorithm modification where the Toffoli gates with the same set of controls are united to form one mEXOR gate.

Step 0. The first row of the truth table consists of the input pattern with the lowest order, $(0, 0, ..., 0)$ which represents the integer 0 as a binary expression. The corresponding output pattern, $(b_1, b_2, ..., b_n)$, unless it consists of all zeros, must be transformed to all zeros. To do so we use one mEXOR gate, $TOF(\emptyset; b_{i_1} + b_{i_2} + ... + b_{i_k})$, where $\{b_{i_1}, b_{i_2}, ..., b_{i_k}\} = \{b_j | b_j = 1, 1 \leq j \leq n\}$.

Step k. The input part of the truth table has the pattern $(a_1, a_2, ..., a_n)$, which represents the binary expansion of the integer k. The output part has the pattern $(b_1, b_2, ..., b_n)$ which, in general, differs from $(a_1, a_2, ..., a_n)$. For any Boolean pattern $(x_1, x_2, ..., x_n)$ we define the set $X^1 = \{x_j | x_j = 1, 1 \leq j \leq n\}$, a pattern consisting of all 1-bits of $(x_1, x_2, ..., x_n)$. In order to bring $(b_1, b_2, ..., b_n)$ to the form $(a_1, a_2, ..., a_n)$, we need at most two mEXOR gates:

1) **Increase ones.** We apply mEXOR gate $TOF(B^1, A^1 \setminus B^1)$ to bring $(b_1, b_2, ..., b_n)$ to the form $(c_1, c_2, ..., c_n) = (a_1 \vee b_1, a_2 \vee b_2, ..., a_n \vee b_n)$; we change the output part of the truth table as dictated by the gate.
2) **Decrease ones.** We apply mEXOR gate $TOF(A^1, C^1 \setminus A^1)$ to bring $(c_1, c_2, ..., c_n)$ to the form $(a_1, a_2, ..., a_n)$; we change the output part of the truth table as dictated by the gate.

Note that during this step, none of the patterns previously put at their places earlier in the truth table are altered:

- $(b_1, b_2, ..., b_n) \succeq (a_1, a_2, ..., a_n)$ since all the patterns in the order less than $(a_1, a_2, ..., a_n)$ are already at their correct places in the upper part of the truth table.
- It follows from the definition of $(c_1, c_2, ..., c_n)$, $(c_1, c_2, ..., c_n) \succeq (b_1, b_2, ..., b_n) \succeq (a_1, a_2, ..., a_n) \Rightarrow (c_1, c_2, ..., c_n) \succeq (a_1, a_2, ..., a_n)$.

Step $2^n - 1$**.** There are no operations at the last step, since if all of the $2^n - 1$ patterns with lower order are in their places, there is automatically only one spot available for the last pattern, $(1, 1, ..., 1)$.

Complexity analysis. An upper bound on the complexity of the presented algorithm's output is given by the formula $2^{n+1}-4$. Since there are 2^n steps, and each requires at most 2 gates to be added to the network, the total cost is $2*2^n$. A more detailed analysis shows that the first step adds at most one gate, the last step never adds a gate, and the step before the last uses at most one gate (similarly to what we had in the first step). Therefore, the complexity decreases to $2^{n+1}-4$. This bound is reachable, and is therefore tight.

Definition 3: The **complexity function** $L(n)$ is the maximal number of gates required to realize a reversible function of n variables with an optimal network.

We have just proved an upper bound $L(n) \leq 2^{n+1}-4$, hence $L(n) \in O(2^n)$. We next prove a lower bound $L(n) \geq C*2^n$ for a positive constant C.

Lemma 1: The number of distinct mEXOR gates with n variables is (3^n-2^n).

Proof: Each of the variables may participate in a gate as a control or target, or may not be a part of a gate. This gives the possibility of 3^n gates. However, among those 3^n some will not contain any target bits, and therefore will not be an mEXOR gate. The number of these will be 2^n (each bit is allowed to be either a control or is not present). Thus, the number of distinct gates is given by the formula (3^n-2^n). ■

It is interesting to note that an mEXOR gate with zero controls is used at most once in the algorithm (Step 0), but the number of them is exponential, namely 2^n. It happens that considering these gates as a set of NOTs does not lead to a change of asymptotic behavior, which is the focus of this subsection.

Theorem 2: $L(n) \geq \frac{2^n}{\ln 3} + o(2^n)$.

Proof: The number of all reversible functions of n variables is $2^n!$ (the number of permutations of 2^n elements). The total number of mEXOR gates is 3^n-2^n. Supposing that by taking all the possible cascades of mEXOR gates we get different functions (which is, of course, not true), the complexity of the hardest to realize function is given by the formula $\log_{(3^n-2^n)}(2^n!)$. Since some cascades built during such a process will give equal functions, the expression $\log_{(3^n-2^n)}(2^n!)$ gives a lower bound for the cost of the most expensive to build reversible function. This expression can be simplified using Stirling's formula to the form $\frac{2^n}{\ln 3} + o(2^n)$. ■

Theorems 1 and 2 together show that the approach using mEXOR gates is asymptotically optimal.

In the above, we assume that the reversible cost of a single mEXOR gate is 1. However, this does not reflect the cost of an mEXOR gate in a real technology, *e.g.* quantum. Considering the quantum cost calculation estimate described in [8]), the quantum cost of a NOT gate is 1, of a CNOT gate 1, of a Toffoli gate 5, of a generalized Toffoli gate with 10 controls 200. For the generalized Toffoli gate its quantum cost has been shown to grow linearly (with an addition of one sink/auxiliary bit) with the number of its controls [1].

The quantum cost of an mEXOR gate is comparable to the quantum cost of the generalized Toffoli gate if the mEXOR gates are built using CNOT gates as shown in Figure 6. Further analysis [7] shows that the quantum cost of an mEXOR gate differs from the cost of the generalized Toffoli gate with the same number of controls only marginally.

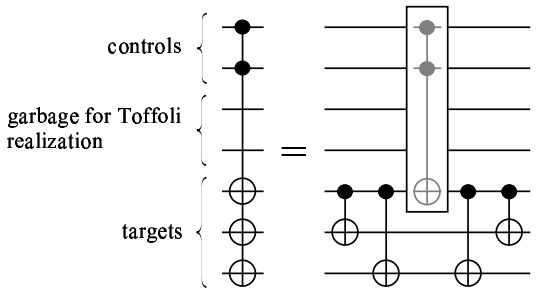

Fig. 6. Construction of a single mEXOR gate

IV. TEMPLATES

In [12], templates were introduced as a tool for network simplification. A template was defined as two sequences of gates that realize the same function. The first sequence of gates is matched to a part of the network being simplified and the second sequence is substituted when a match is found.

In this section, we present an alternative view of templates and give a formal classification which includes the templates used in [12]. To simplify the description of template classes, we adopt the following set notation for used in specifying Toffoli gates [9]:

- C_i represents a set (maybe empty) of lines;
- t_i represents a set containing a single target line;
- All sets are disjoint: $C_i \cap C_j = \emptyset$, $C_i \cap t_k = \emptyset$, $t_l \cap t_k = \emptyset \; \forall i,j,k,l$.

The following result is very useful in defining templates.

Lemma 2: If a network $G_0\ G_1 ...\ G_{m-1}$ 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, $G_1\ G_2 ...\ G_{m-1}\ G_0$. Then all k-shifts can be proven by applying the 1-shift k times. The proof for a 1-shift follows from:

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

■

Now, let a **size** m **template** be a sequence of m gates (a network) that realizes the identity function. Any template of size m must be independent of templates of smaller size, *i.e.* for a given template of size m no application of any set of templates of smaller size can decrease the number of gates or make it equal to another template. The template $G_0\ G_1 ...\ G_{m-1}$ can be applied in two directions:

1) **Forward application:** A series of gates in a network that matches the sequence of gates $G_i\ G_{(i+1) \bmod m} \cdots\ G_{(i+k-1) \bmod m}$ of the template $G_0\ G_1 ...\ G_{m-1}$ exactly, is replaced with the sequence

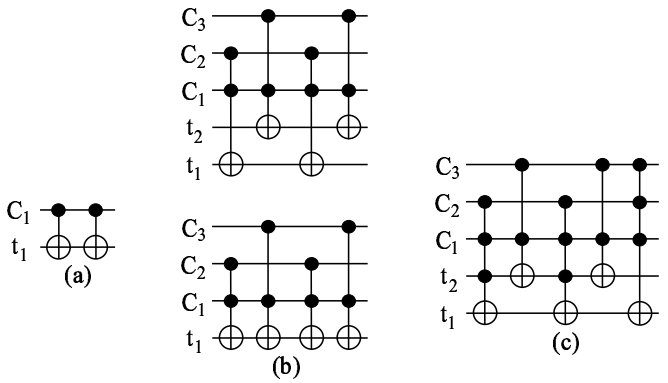

Fig. 7. All templates for $m \leq 5$

$G_{(i-1) \bmod m}\ \ G_{(i-2) \bmod m} \cdots\ \ G_{(i+k) \bmod m}$ without changing the network's output, where $k \in N, k \geq \frac{m}{2}$.

2) **Reverse application:** A series of gates in a 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 the remaining forward applications can be proven using Lemma 2.

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 reverse 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 condition $k \geq \frac{m}{2}$ is used as we do not want to increase the number of gates when a template is applied.

Definition 4: A **class** of templates is defined as a set of templates which can be described by one formula.

The following is a classification of templates up to size 6.

- 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 7a), and it is the **duplication deletion rule** which is described by the sequence $(G_1 G_1)$ where $G_1 = TOF(C_1, t_1)$. When this template is applied, two identical gates are replaced by the empty set which follows since all Toffoli gates are self-inverse.
- m=3. There are no templates of size 3.
- m=4. There is one class of templates (Figure 7b), called the **moving rule**, which can be written as follows $(G_1 G_2 G_1 G_2)$ where $G_1 = TOF(C_1 + C_2, C_4 + C_5)$ and $G_2 = TOF(C_1 + C_3, C_4 + C_6)$. This is called the moving rule because the sequence $G_1 G_2$ is replaced with $G_2 G_1$. This does not reduce the number of gates as in the application of other templates, but rather reorders gates so that other templates can be applied.
  The upper template in Figure 7b 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 one class of template of size 5 (Figure 7c). The class can be written as $(G_1 G_2 G_1 G_2 G_3)$ where $G_1 = TOF(C_1 + C_2 + t_2, t_1)$, $G_2 = TOF(C_1 + C_3, t_2)$ and $G_3 = TOF(C_1 + C_2 + C_3, t_1)$.
- m=6. There are four classes here (Figure 8), and they are described by
  - $G_1 G_2 G_1 G_3 G_4 G_3$ (Figure 8a), where
    $$G_1 = TOF(C_1 + C_3 + t_2, t_1),$$
    $$G_2 = TOF(C_1 + C_2 + C_3 + C_4 + t_1, t_2),$$
    $$G_3 = TOF(C_1 + C_2 + t_1, t_2),$$
    $$G_4 = TOF(C_1 + C_2 + C_3 + C_4 + t_2, t_1).$$
  - $G_1 G_2 G_1 G_3 G_2 G_3$ (Figure 8b), where
    $$G_1 = TOF(C_1 + C_3 + t_2, t_1),$$
    $$G_2 = TOF(C_1 + C_2 + C_3 + C_4 + t_1, t_2),$$
    $$G_3 = TOF(C_1 + C_2 + t_2, t_1).$$
    Note, these two formulas for the classes look very similar, and, in fact using Fredkin gates [4], they can be generalized to form one very simple template: $FRE(C_1 + C_2 + C_3 + C_4, t_1 + t_2)$ $FRE(C_1 + C_2 + C_3 + C_4, t_1 + t_2)$ (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). We do not pursue this here as we are restricting our attention to generalized Toffoli gates.
  - $G_1 G_2 G_1 G_3 G_4 G_2$ (Figure 8c), where
    $$G_1 = TOF(C_1 + C_2 + t_2, t_1),$$
    $$G_2 = TOF(C_1 + C_3 + t_1, t_2),$$
    $$G_3 = TOF(C_1 + C_2 + C_3 + t_1, t_2),$$
    $$G_4 = TOF(C_1 + C_2 + C_3 + t_2, t_1).$$
  - $G_1 G_2 G_3 G_1 G_2 G_3$ (Figure 8d), where
    $$G_1 = TOF(C_1 + C_2 + C_4 + t_2, t_1),$$
    $$G_2 = TOF(C_1 + C_2 + C_3 + t_2, t_1),$$
    $$G_3 = TOF(C_1 + C_3 + C_4 + t_1, t_2).$$

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

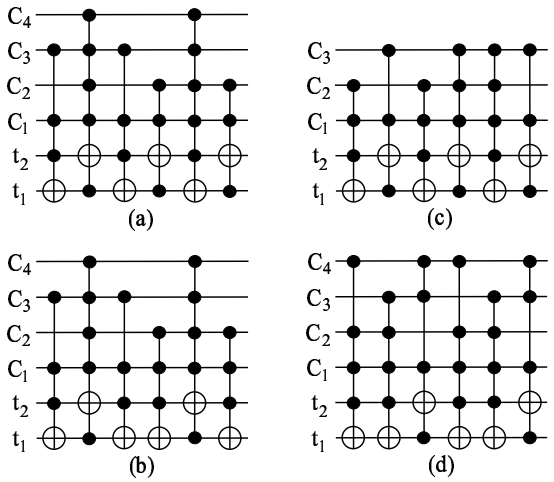

Fig. 8. All templates for $m = 6$

Templates of size 4. The templates are independent of the size 2 template, since no adjacent gates are equal.

Template of size 5.

- The template is independent of the size 2 template, since no adjacent gates are equal.
- The size 4 template can be applied to move gate G_3 anywhere in a template, but it does not allow any simplification of a size 5 template by smaller templates.

Templates of size 6.

- Size 6 templates are independent of the size 2 template, since no adjacent gates are equal.
- A size 4 template can be applied to interchange gates some gates in templates $G_1G_2G_1G_3G_2G_3$ and $G_1G_2G_3G_1G_2G_3$ and does not lead to any simplification.
- The size 5 template matches at most 2 gates of any of the size 6 templates and therefore cannot be applied.

A. Completeness

We have written a program that enumerates all the 4-variable networks with 6 gates that realize the identity function and tries to apply the templates. This program shows that the set of 7 templates given above (G_1G_1, $G_1G_2G_1G_2$, $G_1G_2G_1G_2G_3$, $G_1G_2G_1G_3G_4G_3$, $G_1G_2G_1G_3G_2G_3$, $G_1G_2G_1G_3G_4G_2$, and $G_1G_2G_3G_1G_2G_3$) is the complete set of templates of size 6 or less for 4 inputs and less. We are currently working on finding the templates of size 7 and 8. Finding templates of size 9 does not seem to be feasible at this point.

The mathematical proof of completeness of the presented set of templates for any number of inputs is harder. For templates of size 2 it can be done by inspection. For templates of size 4 and 5 the following lemmas are useful.

Lemma 3: A size m template has at most $\lfloor \frac{m}{2} \rfloor$ different target lines (lines with EXORs).

Proof: Prove by contradiction. Suppose there are $\lfloor \frac{m}{2} \rfloor + 1$ or more lines which contain an EXOR. Then, by the pigeon hole principle, there will be one line with one EXOR only. Cut the cycle so that the gate with this EXOR, $TOF(C, t)$ comes first. Now, if we assign 1 to all $x_j \in C$, the value of t changes to $\bar{t}$ as the signal is propagated in the template. Thus, the template does not realize the identity function, which contradicts its definition. ■

Lemma 4: Given a template of size m where all gates have the same target line, then m is even and all gates can be grouped as pairs of equal gates.

Proof: Proof by contradiction. Suppose not all the gates can be paired or the number m is odd. Apply moving and duplication deletion rules to delete all the paired gates from the template. The remaining network still realizes the identity since all the applied operations did not change the network output. When propagating the signal in the network, the output on the line with the EXOR (for instance, x_n) becomes a polynomial of positive polarity on the remaining variables (for instance, $x_1, x_2, ...x_{n-1}$). In other words, a Zhegalkin polynomial [18] (also knows as a positive polarity Reed-Muller polynomial) on the set of variables $\{x_1, x_2, ...x_{n-1}\}$ was added to the input x_n. Since no non-zero Zhegalkin polynomial equals zero, the output on the n-th line will differ from its input. This contradicts the definition of a template since it realizes the identity function. ■

Use of Lemma 3 allows us to say that all the templates of size 4 have EXORs 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 4. Thus, an exhaustive search proof becomes reasonable. For the size 5 templates we can guarantee that they all will have exactly two lines with EXORs. Note, that Lemma 4 proves that the only two templates which have only one line affected by EXORs are the duplication deletion rule and the moving rule, all other identities of this type are applications of the above rules.

V. Experimental Results

We have written a program which synthesizes a Toffoli network for a given reversible specification and then applies templates to simplify the network achieved on the first step. The synthesis part of the program is straightforward. The simplification part works as follows. First, it is convenient to store template $G_1G_2G_1G_2$ as a separate rule which helps to bring the gates together to match a template. We call this the "moving rule". Then, the network is simplified as follows. For the hierarchy of templates $G_1G_1 \succ G_1G_2G_1G_2G_3 \succ G_1G_2G_1G_3G_4G_3 \succ G_1G_2G_1G_3G_2G_3 \succ G_1G_2G_1G_3G_4G_2 \succ G_1G_2G_3G_1G_2G_3$ (in a sense, more general transformations are applied first), the program tries 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, this is done for the largest number of matched gates possible. After applying a template, the program starts trying to apply the templates in the order given from the beginning. Finally, if none of the templates can be applied, the simplification process stops.

Our current implementation of template matching uses redundant computation that could be eliminated in future implementations. However, for networks with up to 500 gates, the runtime for network simplification using templates is no more than a few seconds, and is negligible for networks with

up to 100 gates. We have thus not worked at optimizing the template matching further as 500 is a practical limit for the networks of current interest. Both simplification and template matching portions of our implementation would have to be optimized to work efficiently for networks with more than 500 gates.

Our template matching tool as implemented does not use any backtracking and applies templates immediately and only to decrease the gate count (or, in modified template matching discussed later, a template can be applied to decrease the number of controls) once a suitable matching is found. This is clearly not optimal, and a more robust template application procedure would produce better results, as was illustrated in [6]. Although we have not investigated the question in detail, optimal network simplification using templates certainly appears to be an NP-hard problem.

Example 1: We took a network for a three bit adder produced by the synthesis algorithm presented in [12] (Figure 9) and applied our template matching procedure to simplify it. As expected [12], the program used a size 5 template and matched 3 gates. Those gates were substituted by the remaining 2 gates of the template read in reverse order. This network is equivalent to the smallest reversible network for a three bit full adder found by hands in earlier papers.

We also investigated a *modified template matching* procedure. It works as follows. Normally, a section of the network 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 network.

Table III shows the results of applying our algorithm to all $8! = 40320$ 3×3 reversible functions. Three scenarios are listed:

(a) the straight forward algorithm with improvements described in [12];
(b) (a) plus template matching;
(c) (a) plus modified template matching;
(d) optimal results from [15].

For each scenario, we show the number of functions for each gate count and the average number of gates required. Runtime for calculating column (c) using an Athlon XP 2400+ computer with 512 Mb of RAM is 35.14 seconds.

TABLE III
NUMBER OF REVERSIBLE FUNCTIONS USING A SPECIFIED NUMBER OF GATES FOR $n = 3$

Size	(a)	(b)	(c)	(d)
17				
16				
15	2			
14	22	8		
13	112	43	6	
12	432	215	62	
11	1191	651	391	
10	2575	1776	1444	
9	5116	4038	3837	
8	7842	7405	7274	577
7	8989	9716	9965	10253
6	7478	8573	9086	17049
5	4314	5167	5448	8921
4	1682	2055	2125	2780
3	463	558	567	625
2	89	102	102	102
1	12	12	12	12
0	1	1	1	1
WA:	7.25	6.92	6.80	5.87

(a): before template matching
(b): after template matching
(c): modified template matching
(d): optimal sizes [15]

VI. BENCHMARKS

An irreversible function can be realized using reversible gates [14]. Garbage outputs must be added if necessary so that the output patterns are distinct and constant inputs must be added as necessary so that the function has the same number of inputs and outputs. This can be viewed as embedding the irreversible function specification in a larger reversible one.

Definition 5: The maximum output pattern **multiplicity** of a multiple-output Boolean function is the maximum number of input assignments which yield the same output pattern. Equivalently, it is the maximum number of times a single output pattern appears in the truth table specification of the function.

As shown in [10], the minimum number of garbage outputs required is $\lceil \log_2 q \rceil$, where q is the maximum output pattern multiplicity of the irreversible function. Optimal definition of the garbage outputs is a difficult and open problem. At present we pre-assign them to make the function reversible using the approach described in [11]. Often they can simply be set equal to input variables. At other times, we use EXOR functions involving subsets of the inputs.

Table IV shows our results for a number of benchmark functions. The **name**, **in** and **out** columns represent the name of a benchmark function, the number of inputs and outputs. **Size** is the number of inputs and outputs in a minimal reversible specification (Theorem 1 of [10]) derived from the benchmark. The **cost before** and **cost after** columns contain the number of generalized Toffoli gates needed before and after the template matching tool is applied. For the presented networks the gate count reduction resulting from template matching is not significant. There are a number of reasons for this.

All but the first two benchmarks have inherent structure. The synthesis phase of our approach is in fact doing a good job of finding that structure. In addition, as noted above our approach to applying templates is greedy and may well miss possible specifications. This is not to say templates are not of use since as shown above over all 3 variable functions they actually provide significant reduction. In addition as noted earlier, templates can reduce the number of control connections if not the gate count, reducing the implementation cost in certain technologies.

Simple gate count gives an estimate but it is often more informative to evaluate cost in terms of a particular technology.

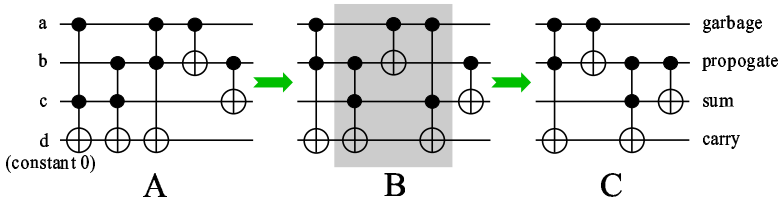

Fig. 9. Optimal network for a full adder

TABLE IV
RESULTS OF BENCHMARK FUNCTION SYNTHESIS

name	in	out	size	cost before	cost after	QC	TG QC	TG QC +1G
3_17	3	3	3	6	6	12	5	5
4_49	4	4	4	16	16	58	13	13
add3	3	2	4	5	4	8	13	13
ham3	3	3	3	6	5	7	5	5
ham7	7	7	7	25	23	81	125	84
ham15	15	15	15	138	132	2084	32765	352
hwb4	4	4	4	18	17	63	13	13
hwb5	5	5	5	57	55	313	29	29
hwb6	6	6	6	134	126	1528	61	52
hwb7	7	7	7	302	289	5419	125	84
4mod5	4	1	5	9	8	24	29	29
5mod5	5	1	6	18	17	185	61	52
rd53	5	3	7	12	12	132	125	84
cycle17_3	20	20	20	48	48	7802	1048573	512

For that reason, we calculate the quantum cost[1] of our networks based on the reported quantum costs of Toffoli gates [1], [8]. Column **QC** in Table IV provides the quantum cost of each network assuming the network is built with the number of lines given in the **size** column (that is, no additional lines are used). Column **TG QC** gives for comparison the quantum cost of a single size n Toffoli gate built on n lines, given n equals to the size of the corresponding function. The last column, **TG QC +1G**, contains the cost of a size n Toffoli gate built on $(n+1)$ lines (that is, one additional line is available). Even though we report calculation of the quantum costs, we point out that the primary goal of the presented work was minimization of the gate count, not quantum cost minimization. We also have some evidence that using more gates with less controls will decrease the quantum cost for some of the presented problems. The actual designs of the reported networks can be viewed online at http://www.cs.uvic.ca/˜dmaslov/.

The reversible specifications for the functions $ham3$, $ham7$, $ham15$, (hamming optimal coding functions, where the number defines the size) $hwb4$, $hwb5$, $hwb6$, $hwb7$ (hidden weighted bit functions, where the input pattern is circularly shifted by the number of ones it has) and $4mod5$ (Grover oracle which interprets the 4-bit input as a binary expansion and returns 1 if the number given by this 4-bit string is divisible by 5) were sent to us by Patel and Markov.

The maximal size function for which we reported a network has 20 inputs and outputs. This function, $cycle17\_3$, cycles the first 2^{17} input values according to the binary value of the last three bits that act as a control. In other words, if the binary number represented by the controls is A, the input set $\{0, 1, 2, ..., 2^{17} - 1\}$ is mapped into $\{A, A + 1, ..., 2^{17} - 1, 0, 1, ..., A - 1\}$. $Cycle17\_3$ can also be thought of as a modula-2^{17} adder which adds 17-bit and 3-bit numbers. The network structure is surprisingly regular, Figure 10. One easily sees how to build analogous networks for general type shifters $cycleN\_K$ (parameters N, K are natural numbers). For instance, the network for $cycle10\_10$ (modula-1024 adder of two 10-bit numbers) will have 55 gates. Using an Athlon XP 2400+ computer with 512 Mb of RAM it took approximately 25 minutes to synthesize the network for $cycle17\_3$ (runtimes for smaller specifications are noticeably smaller, *e.g.* 8.9 second for the second largest function that has size 15).

[1]We estimate the quantum cost as the number of 1-qubit or two qubit controlled-V operations needed to construct the gate (see [1], [14] for more information). This is an approximation of the actual cost, but it gives a closer estimate of the real cost than does simple gate count.

VII. CONCLUSION

A simple algorithm for the synthesis of reversible networks composed of generalized Toffoli gates has been presented. The basic algorithm will always terminate with a valid network. An heuristic approach has been given to reduce the size of the networks produced through Toffoli gate control line reduction. The major enhancement to the basic algorithm is a method by which gates can be identified at either end of the specification and the network synthesized in both directions simultaneously. A modification of this algorithm synthesizes asymptotically optimal reversible networks of mEXOR elements.

In this work we also described a template simplification tool, a tool which allows further reduction of the networks produced by the synthesis algorithm. Templates were defined, classified for simplicity of their application and presented in a readily usable form.

We wrote a program and tested the synthesis algorithm, template application tandem. The first test, exhaustive synthesis

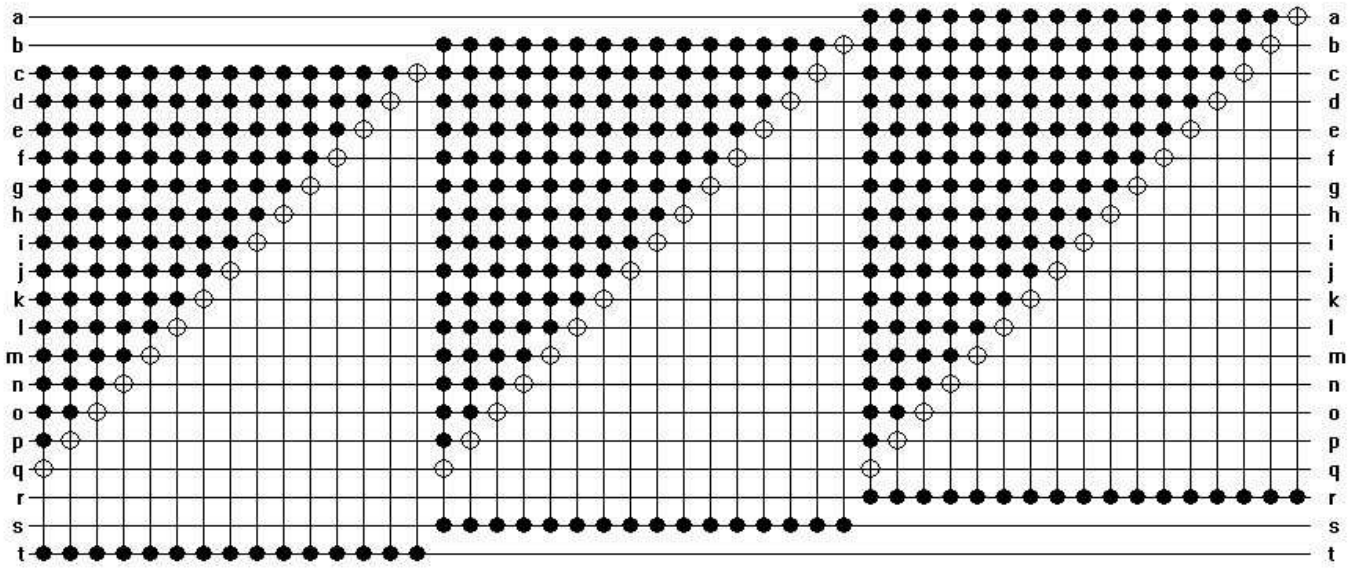

Fig. 10. Network for $cycle17\_3$ function

of all 3-variable reversible functions showed that more than 50% of synthesized functions are optimal, which shows the quality of our synthesis approach. Next we ran our program to synthesize benchmark functions and achieved good results.

There are several open areas of research arising from this work. We are considering nonexhaustive ways to identify larger templates and also looking at better ways to apply template matching.

Perhaps the most interesting open question is how to effectively assign garbage outputs. We are currently developing a modification to our approach that treats the garbage outputs as don't-cares throughout the synthesis procedure.

Finally, this paper has considered only Toffoli gates. The approaches described are readily extended to other reversible gates *e.g.* Fredkin gates. Preliminary study shows that this can result in significantly reduced gate count.

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. *Reversible Logic Synthesis*. PhD thesis, University of New Brunswick, Fredericton, Canada, October 2003.

[7] 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.

[8] D. Maslov and G. Dueck. Improved quantum cost for n-bit Toffoli gates. *IEE Electronics Letters*, 39(25):1790–1791, December 2003. Corrected and expanded version: quant-ph/0403053.

[9] D. Maslov, G. Dueck, and M.Miller. Simplification of Toffoli networks via templates. In *Symposium on Integrated Circuits and System Design*, pages 53–58, September 2003.

[10] 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.

[11] 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*, pages 56–62, March 2003.

[12] 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.

[13] A. Mishchenko and M. Perkowski. Logic synthesis of reversible wave cascades. In *International Workshop on Logic Synthesis*, pages 197–202, June 2002.

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

[15] 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.

[16] V.V. Shende, A.K. Prasad, I.L. Markov, and J.P. Hayes. Synthesis of reversible logic circuits. *IEEE Transactions on CAD*, 22(6):723–729, June 2003.

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

[18] I. I. Zhegalkin. The technique of calculation of statements in symbolic logic. *Matematichesky Sbornic*, 34:9–28, 1927. (In Russian).

Dmitri Maslov received the M.S. in mathematics degree from Lomonosov's Moscow State University, Russia, in 1999, and the M.S. and Ph.D. in computer science from the University of New Brunswick, Canada, in 2002 and 2003, respectively.

He is a postdoctoral fellow in the Department of Computer Science, University of Victoria, Canada. His current research interests include reversible logic and its synthesis, quantum computations, EXOR minimization, and synthesis of secure cryptographic hardware.

Gerhard W. Dueck (S83-M89) was born in Montevideo, Uruguay. He received the B.Sc., Master, and Ph.D. degrees in computer science for the University of Manitoba, Winnipeg, Manitoba, Canada, in 1983, 1986, and 1988, respectively.

He is currently a professor in the Faculty of Computer at the University of New Brunswick. After completing his PhD he joined St. Francis Xavier University in Antigonish, Nova Scotia. In 1991 he spent a year at the Naval Postgraduate School in Monterey, CA, as a research associate. In 1999 he joined the Faculty of Computer Science at the University of New Brunswick. He has been actively involved in the IEEE Computer Society Technical Committee on Multiple-Valued Logic, where he severed as chair in 1998 and 1999. He was program chair of the IEEE International Symposium on Multiple-Valued Logic in 1993 and 2004 and symposium chair in 1997. His research interests include reversible logic, Reed Muller expansions, multiple-valued logic, and digital design.

D, Michael Miller (M'85) received the B.Sc. degree in physics and mathematics from the University of Winnipeg in 1971 and the M.Sc. and Ph.D. degrees in computer science from the University of Manitoba in 1973 and 1976 respectively.

Dr. Miller was a faculty member at the Universities of New Brunswick, Winnipeg and Manitoba, before joining the University of Victoria as Chair of the Computer Science Department in 1987. Dr. Miller is currently Dean of the Faculty of Engineering at the University of Victoria, a position he has held since 1997.

Dr. Miller's research interests are in decision diagrams, reversible and quantum logic, spectral logic and multiple-valued logic. He is a co-author of two books on spectral logic and has published over 100 papers in his areas of interest.

Dr. Miller is the Secretary for the IEEE Computer Society Technical Committee on Multiple-Valued Logic.