Reversible Logic Synthesis with Fredkin and Peres Gates

JAMES DONALD and NIRAJ K. JHA
Princeton University

Reversible logic has applications in low-power computing and quantum computing. Most reversible logic synthesis methods are tied to particular gate types, and cannot synthesize large functions. This article extends RMRLS, a reversible logic synthesis tool, to include additional gate types. While classic RMRLS can synthesize functions using NOT, CNOT, and n-bit Toffoli gates, our work details the inclusion of n-bit Fredkin and Peres gates. We find that these additional gates reduce the average gate count for three-variable functions from 6.10 to 4.56, and improve the synthesis results of many larger functions, both in terms of gate count and quantum cost.

Categories and Subject Descriptors: B.6.3 [**Logic Design**]: Design Aids—*Automatic synthesis*

General Terms: Design

Additional Key Words and Phrases: Quantum computing, reversible logic

ACM Reference Format:

Donald, J. and Jha, N. K. 2008. Reversible logic synthesis with Fredkin and Peres gates. ACM J. Emerg. Technol. Comput. Syst. 4, 1, Article 2 (March 2008), 19 pages. DOI = 10.1145/1330521.1330523 http://doi.acm.org/10.1145/1330521.1330523

1. INTRODUCTION

Reversible logic is motivated by its applications in low-power computing. Landauer's principle says that some finite amount of energy will be lost for any irreversible computation, but this can be avoided in a fully reversible logic implementation [Landauer 1961]. The challenges of managing power density in modern and future electronics is a strong reason for seeking low-power techniques such as reversible logic [de Vos 1994]. Furthermore, reversible logic has applications in communication [Smolin and DiVincenzo 1996], optical computing [Cuykendall and Andersen 1987], biosynthesis of messenger RNA [Bennett 1973], and particularly quantum computing [Barenco et al. 1995]. In order for such reversible logic technologies to be feasibly implemented, full

This work was supported in part by the NSF under Grant No. CCF-0429745.
Author's address: J. Donald (corresponding author), N. K. Jha, Department of Electrical Engineering, Princeton University, Princeton, NJ 08544; email: jdonald@princeton.edu

design-flow methodologies, including logic synthesis, must be developed as in the case of conventional irreversible circuits [Perkowski et al. 2001].

In Gupta et al. [2006] the authors presented an algorithm for reversible logic synthesis using networks of n-bit Toffoli gates. Their algorithm works by searching for candidate factors in the positive polarity Reed-Muller (PPRM) forms of the representative equations that can be substituted to form the operation of a NOT, CNOT, or n-bit Toffoli gate. By traversing a search tree along with practical pruning, their algorithm is capable of synthesizing a wide range of reversible functions with as many as 16 variables.

An n-bit Toffoli gate can be thought of as a controlled XOR operation, and is defined as follows [Toffoli 1980].

$$\begin{aligned} y_i &= x_i \quad \text{for} \quad 1 \leq i \leq n-1 \\ y_n &= x_n \oplus x_1 x_2 \dots x_{n-1} \end{aligned} \tag{1}$$

A CNOT gate can be thought of as a two-bit Toffoli gate, and a one-bit Toffoli gate ($y_1 = x_1 \oplus 1$) is the same as a NOT gate. In quantum computing, the NOT, CNOT, and three-bit Toffoli gates are known to have quantum costs of 1, 1, and 5, respectively, while Toffoli gates of four or more bits have even greater quantum costs [Barenco et al. 1995]. Another popular gate is the n-bit Fredkin gate, defined as follows [Fredkin and Toffoli 1982].

$$\begin{aligned} y_i &= x_i \quad \text{for} \quad 1 \leq i \leq n-2 \\ y_{n-1} &= x_{n-1}\overline{x_1 x_2 \dots x_{n-2}} + x_n x_1 x_2 \dots x_{n-2} \\ y_n &= x_n \overline{x_1 x_2 \dots x_{n-2}} + x_{n-1} x_1 x_2 \dots x_{n-2} \end{aligned} \tag{2}$$

For $n = 2$, a two-bit Fredkin gate can be thought of as an unconditional SWAP gate (i.e., $y_1 = x_2$ and $y_2 = x_1$). A two-bit SWAP gate has a quantum cost of 3 and a three-bit Fredkin gate has a cost of 5 [Smolin and DiVincenzo 1996], while higher-order Fredkin gates are even more expensive [Maslov et al. 2007]. A third type of gate considered in this article is the three-bit Peres gate [Peres 1985]. A Peres gate can simultaneously accomplish the operation of both a CNOT gate and a three-bit Toffoli gate, with an operation defined as follows.

$$\begin{aligned} y_1 &= x_1 \oplus x_2 \\ y_2 &= x_2 \\ y_3 &= x_3 \oplus x_1 x_2 \end{aligned} \tag{3}$$

Although the definition of Peres gates given in Eq. (3) can be extended to include multiple control bits and thus n-bit Peres gates, this work primarily deals with three-bit Peres gates to remain consistent with the most common definition used by the reversible logic community. The three-bit Peres gate is known to have a quantum cost of 4 [Hung et al. 2004].

Because the set of NOT, CNOT, and three-bit Toffoli gates is known to be capable of synthesizing any reversible function, many existing reversible synthesis algorithms do not yet attempt to include "optional" gates such as Fredkin and Peres.

In addition to Gupta et al. [2006] there have been many recent works on reversible logic synthesis, several of which provide synthesis algorithms that can be practically used, at least for reversible functions of a limited number of variables. Iwama et al. [2002] presented a framework for synthesis through repeated local transformations. Shende et al. [2003] implemented an algorithm to find optimal circuits and provided their corresponding proofs of constructability, although their method is limited to functions of at most a few variables. Maslov and Dueck [2004] proposed heuristic methods to synthesize reversible functions, with the aim of minimizing garbage output. They further demonstrated the use of template matching as a heuristic technique to take suboptimal circuits and simplify them with a template library [Maslov and Dueck 2005]. This has even been extended to include Fredkin gates [Maslov et al. 2003]. However, their template-based technique generates less efficient circuits compared to Gupta et al. [2006] and the addition of new gate types requires the generation of an entirely new template library. Miller et al. [2003] demonstrated an algorithm to synthesize reversible functions using various bidirectional transformations, although this algorithm often generates quite suboptimal solutions even for functions of as few as three variables.

Our work chooses to build upon RMRLS (Reed-Muller reversible logic synthesizer) [Gupta et al. 2006] because this algorithm has been shown to have speed, success rate, and circuit minimization abilities often exceeding those of the aforementioned algorithms. We further believe this to be a more robust and extensible platform for parameterizable synthesis options, unlike algorithms requiring prederived template libraries, such as those in Maslov et al. [2003]. This work implements and evaluates extensions to RMRLS for synthesizing circuits with additional gate types. Our specific contributions are as follows.

—We propose and implement extensions to the RMRLS algorithm to include n-bit Fredkin gates and Peres gates, and detail our methodology that can be applied to any fundamental reversible logic gate.
—We show that going from the NOT, CNOT, and n-bit Toffoli gate (NCT) library to the additional SWAP, Fredkin, and Peres gates (NCTSFP) library reduces the average gate count for three-variable functions from 6.10 to 4.56.
—We synthesize all of the special-purpose reversible functions from Gupta et al. [2006] in addition to some functions from Maslov et al. [2007] that could not be synthesized with classic RMRLS, and show that the additional gate types can reduce the gate counts and quantum costs of synthesized circuits.

The rest of the article is organized as follows. Section 2 provides motivational examples for including SWAP, n-bit Fredkin, or Peres gates in the synthesis algorithm. Section 3 provides our methodology for extending RMRLS. Section 4 details our synthesis of numerous reversible functions and compares these results to those of existing NCT RMRLS as well as other synthesis algorithms. Section 5 offers our conclusions.

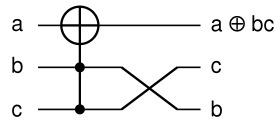

Fig. 1. Two-gate implementation of the reversible specification given in Eq. (4) using a Toffoli and a SWAP gate.

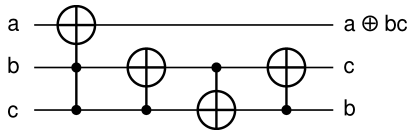

Fig. 2. Four-gate implementation of the reversible specification given in Eq. (4) using Toffoli and CNOT gates.

2. MOTIVATIONAL EXAMPLES

This section motivates the need for the support of Fredkin and Peres gates in reversible logic synthesis. We show some simple functions that can benefit from synthesis with the additional gate types, with the suggestion that larger functions are also likely to benefit, as evidenced by the experimental results given later.

The simplest kind of Fredkin gate is a two-bit Fredkin gate, also known as a SWAP gate. For an example of a function that might use this gate in synthesis, consider

$$\begin{aligned} a_{out} &= a \oplus bc \\ b_{out} &= c \\ c_{out} &= b. \end{aligned} \tag{4}$$

This function quite apparently contains a swap between the variables b and c. A two-gate realization for this function is shown in Figure 1. Alternatively, when using only Toffoli and CNOT gates, a minimum realization with four gates is shown in Figure 2. Being able to accomplish the swap operation with a single gate has advantages in the synthesis process. In the RMRLS algorithm, for example, performing a swap using three CNOT gates might be a solution that is found only after branching in many other failed directions. An algorithm that can properly identify and move forward on swap opportunities may have a better chance at quickly obtaining a solution.

Our next example presents a function that may be realized with the use of a three-bit Fredkin gate. The Boolean equivalents of the operations of n-bit Fredkin gates may be less obvious when $n > 2$. For example, suppose we need to realize the function

$$\begin{aligned} a_{out} &= a \oplus bc \\ b_{out} &= b \oplus ab \oplus ac \\ c_{out} &= c \oplus ab \oplus ac. \end{aligned} \tag{5}$$

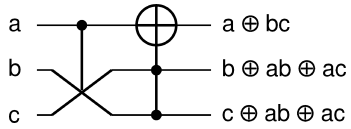

Fig. 3. Two-gate implementation using a three-bit Fredkin gate.

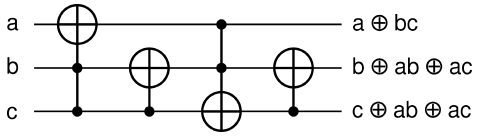

Fig. 4. A practical four-gate implementation of the reversible specification given in Eq. (5) using only Toffoli and CNOT gates.

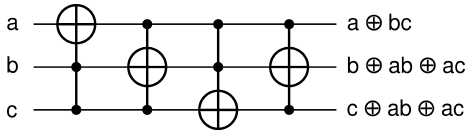

Fig. 5. Suboptimal four-gate implementation of the reversible specification given in Eq. (5) using only Toffoli gates, as can best be found by NCT RMRLS.

To see how this function could be synthesized using a controlled swap (i.e., a Fredkin gate), it is easiest to look at one such realization as shown in Figure 3. A four-gate realization without using Fredkin gates is shown in Figure 4. Even for n-bit Fredkin gates whose Boolean operations may be less intuitive, identifying the single-gate substitutions may improve the search algorithm's synthesis ability and also reduce the average gate count.

An additional problem arises when attempting to synthesize the aforesaid function using RMRLS. The circuit obtained by classic RMRLS is shown in Figure 5. Although this circuit also consists of four gates, it uses more control bits. If the quantum cost [Maslov 2003] for these circuits is evaluated, the cost of the circuit in Figure 4 is 12 while that of the circuit in Figure 5 is 20. Although RMRLS has been shown effective at reducing the gate count, its search techniques do not necessarily minimize the number of control bits, and hence quantum cost, very well.

These examples of circuit simplification provide motivation for using SWAP or n-bit Fredkin gates. A similar argument can be made for including the Peres gate in synthesis, as it is expected to often take the place of a CNOT gate and three-bit Toffoli gate and thus slightly reduce the gate count. Figure 6 shows a possible implementation of Eq. (5) using a Peres gate, Toffoli gate, and CNOT gate. Its quantum cost is only 10.

The examples given in this section show that early detection and placement of Fredkin and Peres gates enables the possibility of improved synthesis results. The following section details our methodology for the modifications that allow RMRLS to implement these additional gate types.

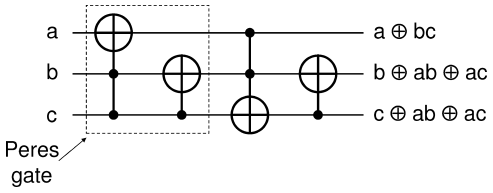

Fig. 6. Three-gate implementation of the reversible specification given in Eq. (5) using a Peres gate.

3. METHODOLOGY

We next present our synthesis methodology.

3.1 Fredkin Candidate Factors

The synthesis technique in this article is an extension of RMRLS [Gupta et al. 2006]. RMRLS uses the PPRM expansion of reversible functions and then traverses a search tree to synthesize a circuit through matching candidate factors. These factors refer to common subexpressions in the Reed-Muller expansions. What ties the original algorithm to Toffoli gates is only that the candidate factors and corresponding substitutions are designed to match the operation of Toffoli gates. In order to extend the algorithm to Fredkin gates, it is necessary to devise a similar scheme to detect Fredkin candidate factors.

In the original NCT-enabled RMRLS, the candidate factors for Toffoli gate transitions must be of the form

$$v_{out,i} = v_i \oplus factor \oplus \ldots, \tag{6}$$

where v_i refers to a single variable, $v_{out,i}$ refers to the unique output line labeled by v_i, and *factor* is a term that does not contain v_i. There may also be other terms in the PPRM expression for $v_{out,i}$.

The corresponding substitution performed if such a candidate factor is found mimics the operation of a Toffoli gate, and thus looks similar to the expression given in Eq. (6).

$$v_i \rightarrow v_i \oplus factor \tag{7}$$

In fact, a relaxed rule allows the substitution to be performed even in the case when v_i does not appear in the original expression. Any sort of relaxed detection schemes are legal, but the substitution itself must be carried out in its entirety as without omitting v_i. For example, even if a Toffoli gate substitution is initiated from just the PPRM expression $a_{out} = bc$, the substitution performed would be $a \rightarrow a \oplus bc$.

Just as Toffoli candidate factors are Boolean expressions revealing the operation of Toffoli gates, the form of a Fredkin candidate factor can be obtained by looking at the definition of the operation of a Fredkin gate. This requires rewriting the definition from Eq. (2) in PPRM form. Since the expression must be in positive polarity, there can be no complement operations, and all intermediate

operations must be either AND or XOR. Thus the expressions become

$$\begin{aligned} y_{n-1} &= x_{n-1} \oplus x_{n-1}(x_1x_2\ldots x_{n-2}) \oplus x_n(x_1x_2\ldots x_{n-2}) \\ y_n &= x_n \oplus x_{n-1}(x_1x_2\ldots x_{n-2}) \oplus x_n(x_1x_2\ldots x_{n-2}). \end{aligned} \tag{8}$$

When we represent the common terms as a characteristic *factor*, this becomes

$$\begin{aligned} v_{out,i} &= v_i \oplus v_i(factor) \oplus v_j(factor) \\ v_{out,j} &= v_j \oplus v_i(factor) \oplus v_j(factor). \end{aligned} \tag{9}$$

This is more restrictive than the candidate factors that give rise to Toffoli substitutions. First, it requires a total of six matching terms. Second, it requires the terms in a PPRM expression for $v_{out,i}$ to match terms in another PPRM expression for $v_{out,j}$. As a result, these expressions are not a subset of those with Toffoli candidate factors, but rather comprise a disjoint set.

Just as the Toffoli conditions have a relaxed form, we also opt to use relaxed rules for Fredkin conditions so as not to require all six matching terms. One advantage of a relaxed approach is simplification of the candidate factor detection scheme. Another advantage is that there may still be a benefit from using a Fredkin gate, even when not all six terms are present. If all six terms are present, use of a Fredkin gate can potentially reduce the six terms to only two. If one of the six terms is missing, for example, we could likely end up with three terms remaining, whereas the substitution of a Toffoli gate is typically made with the expectation of reducing the complexity by one term. If the result of a substitution turns out to be ineffective, it will automatically be given a low priority in the search tree. The downside of such relaxed conditions is that many of these extra nodes, along with their children, can pollute the priority queue and slow down the algorithm.

In the end, we chose to use a candidate factor requirement needing only two matching terms. The requirement is that a PPRM expression for $v_{out,i}$ must contain terms $v_i(factor)$ and $v_j(factor)$. This simplifies the search procedure, since such a restriction does not depend on terms in the PPRM expression of $v_{out,j}$. Under this condition alone, a potential Fredkin branch in the search tree can be created with as little as two out of the six of ideal matching terms. Our search procedure mirrors that of the search for Toffoli candidate factors. In the Toffoli case, we would search for terms in $v_{out,i}$ that do not contain v_i. For the Fredkin case, we instead examine any terms that do contain v_i. Once these are found, we search for any matching subfactor, in the same PPRM expression, that is identical to the candidate factor, except for a replacement of v_i with some other variable v_j. For example, the PPRM expression $c_{out} = abc \oplus abd$ satisfies this condition because abc does contain c and abd is the corresponding subfactor with c replaced by d.

Among special cases, assuming *factor* to be 0 in Eq. (9) results in mere pass-through gates, or gates that do nothing. Detecting and substituting in such expressions is useless. When $factor = 1$, however, this is a candidate for a direct SWAP (two-bit Fredkin). This case must be detected differently from typical n-bit Fredkin cases where $n \geq 3$. This can be seen by substituting $factor = 1$ into the expressions and seeing that the $v_i(factor)$ term cancels out and thus cannot

be detected directly. Thus, the two-bit swap is somewhat of an exception in the candidate factor detection scheme. The requirement for a matching candidate factor is that of a term containing only a single variable that is different from $v_{out,i}$. Again, this is one form of a relaxed rule. A more restrictive requirement would be that $v_{out,i}$ contains v_j while $v_{out,j}$ contains v_i (e.g., $a_{out} = b$ and $b_{out} = a$). However, we opt for the less restrictive case because it gives us the most room for exploration.

3.2 Fredkin Substitutions

Appropriate substitutions are performed upon locating suitable candidate factors and their matching subfactors. These substitutions match the operation of a Fredkin gate as shown earlier, and are specified as follows.

$$\begin{aligned} v_i &\rightarrow v_i \oplus v_i(factor) \oplus v_j(factor) \\ v_j &\rightarrow v_j \oplus v_i(factor) \oplus v_j(factor) \end{aligned} \tag{10}$$

There is one additional complexity in the implementation of complex substitutions such as the one required for a Fredkin substitution. For the Toffoli substitutions, we have the advantage that *factor* would never contain its corresponding v_i. One convenient effect of this is that when a PPRM expression is represented as a sorted linked list, to perform the substitution, the appropriate terms can be added without the worry that they may affect later substitutions. When performing a Fredkin substitution or even a two-way SWAP, we do not have this luxury since the additional terms will contain instances of v_i and v_j. In order to work around this, we do not insert new terms into the linked list "in place." Rather, we create an entirely new linked list of PPRM terms, then destroy the old one. This implementation issue likely adds some overhead, although its runtime is at most linear with respect to the length of the linked list of PPRM terms.

3.3 Peres Candidate Factors

The Peres gate presented in Section 1 must also have its corresponding candidate factors and substitutions in order to be implemented in synthesis. Since a Peres gate is equivalent to a three-bit Toffoli gate followed by a CNOT gate, we in fact use the same candidate factor search mechanism as already implemented. Since we restrict our study to three-bit Peres gates, we add the artificial restriction that the *factor* which does not contain v_i must consist of exactly two variables.

Unlike Toffoli and Fredkin gates, the Peres gate is not self-reversible. This means that the reverse of the Peres gate [Peres 1985] is actually a different gate, and we must account for this in our synthesis. The PPRM functional specification of the reverse-Peres gate is shown next.

$$\begin{aligned} y_1 &= x_1 \oplus x_2 \\ y_2 &= x_2 \\ y_3 &= x_2 \oplus x_3 \oplus x_1x_2 \end{aligned} \tag{11}$$

In all, the only difference between this specification and that of the Peres gate is the additional x_2 term in the PPRM expression for y_3. Thus, we can use this as the distinguishing choice to decide between whether to apply a Peres or reverse-Peres gate upon encountering the appropriate two-variable candidate factor. Suppose a PPRM contains a Toffoli candidate factor as specified in Eq. (6).

$$v_{out,i} = v_i \oplus factor \oplus \ldots$$

To be considered as a Peres candidate, the *factor* term must contain two variables, say a and b. If a also appears in the PPRM expression for $v_{out,i}$, a search branch is attempted using a reverse-Peres gate with a as the control bit. If b also appears in the PPRM expression for $v_{out,i}$, a search branch is attempted using a reverse-Peres gate with b as the control bit. If either a or b does not appear in the PPRM expression, then branches would be created using a regular Peres gate with a or b as the control bit.

As with preceding relaxed candidate factor detection schemes, the Peres detection schemes do not require v_i to appear in the PPRM for $v_{out,i}$. Even if many of these substitutions turn out to be poor choices, the optimistic approach here assumes that poor substitutions will be properly tagged as such in the search algorithm.

3.4 Peres Substitutions

The non-self-reversibility of Peres gates brings up another issue when considering the appropriate substitutions to apply upon matching candidate factors. To properly transform the PPRM expressions, we find that the substitutions on applying a Peres gate turn out to be the exact substitutions that define the operation of a reverse-Peres gate. The substitutions applied are

$$\begin{aligned} v_i &\rightarrow b \oplus v_i \oplus factor \\ a &\rightarrow a \oplus b, \end{aligned} \tag{12}$$

where *factor* refers to the two-variable expression, b refers to one of two variables chosen to be the control bit, while a is the other variable. Thus $factor = ab$. For the aforesaid transformations, it may be tempting to serialize the substitutions as $a \rightarrow a \oplus b$ followed by $v_i \rightarrow v_i \oplus factor$. Although this implementation leads to a correct result while the other possible serialization does not, it is best to reason about and implement all multiple substitution rules as simultaneous substitutions. This is especially important in Fredkin gate substitutions, as no possible serialization for Eq. (10) can correctly perform a controlled swap.

Because the substitution required for a reverse-Peres gate reflects the operation of a Peres gate, its expressions are slightly simpler, matching the equations defining the operation of a forward-Peres gate given in Eq. (3).

$$\begin{aligned} v_i &\rightarrow v_i \oplus factor \\ a &\rightarrow a \oplus b \end{aligned} \tag{13}$$

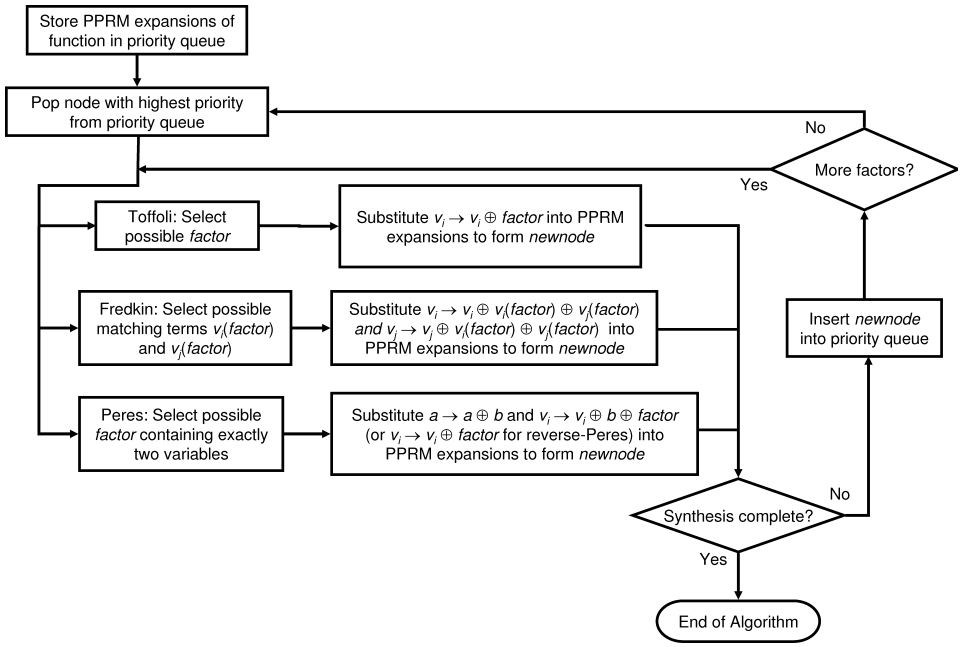

Fig. 7. Flow chart depicting the search procedure in RMRLS with multiple gate types.

In Section 3.6, we provide further intuition for why Peres gate substitutions are defined by the reverse-Peres gate equations, and explain how this duality applies to other gates.

As in the implementation of Fredkin substitutions, the modification of the PPRM linked lists requires creating an entirely new list. The optimization of an "in-place" linked list modification is probably only applicable for the special case of Toffoli gates.

3.5 Algorithm

The encompassing algorithm framework is still the same as the one used in the original NCT-enabled RMRLS [Gupta et al. 2006]. A priority queue is used for storing partial solutions, and at each iteration a new node is explored. The differences between our algorithm and the original one reside in the two new kinds of candidate factor detection schemes and their corresponding substitutions. A flow chart depicting a high-level view of the algorithm is given in Figure 7. This diagram is in the form of the algorithm given in Agrawal and Jha [2004], although it has been extended to include all three kinds of candidate factor detections and their corresponding substitutions.

Our method for adding new gates into the RMRLS framework suggests that synthesis with any other primitive gate types, such as Miller [2002] and Margolus [1988] gates, could also be developed using the same basic approach. The two primary requirements for enabling synthesis of any arbitrary gate are a candidate factor condition and a procedure for performing the gate's substitutions. Candidate factor conditions can be obtained by writing the gate's

functional specification in PPRM form. The substitution used for such a branch can be derived by writing the functional specification of the gate's reverse in PPRM form. The reasons for this slight technicality with non-self-reversible gates is explained in Section 3.6.

The flow given in Figure 7 actually only considers the case of an exhaustive search where the algorithm terminates once a solution is found. There are actually other customizable options, such as a greedy heuristic search and the ability to continue searching for better solutions after one solution is found.

RMRLS also contains many heuristic parameters to tune its priority queue mechanism and pruning limits. Although we also believe that these heuristics may have potential for dramatically improving the algorithm's performance and success rate, we opted not to modify these configurations. This way we were able to provide fair comparisons against the NCT results provided in Gupta et al. [2006].

3.6 Reversed Substitutions

Because RMRLS synthesizes reversible circuits in the forward direction, it may not be obvious as to why the substitutions for a Peres gate reflect the operation of a reverse-Peres gate. Because NOT, CNOT, and Toffoli gates are self-reversible, this directionality was not an issue in the original RMRLS.

Although RMRLS adds new gates from the beginning to the end of a synthesized circuit, its starting point is the target function and its ending point the identity function. Thus, a reversal effect is achieved while still applying gates in forward order. This comes about because of a duality between substitution and operation. An operation, for example, $a_{out} \leftarrow b_{out} \oplus c_{out}$, sets a_{out} based on the current PPRM expressions for other output variables. A substitution, for example, $a \rightarrow b \oplus c$, modifies any and all PPRM expressions that contain a. An operation has the effect of placing a new gate at the output, while a substitution has the effect of placing a new gate at the *input* of the currently expressed function.

Because each substitution has the purpose of gradually simplifying the original function toward the identity function, the proper transformations must actually represent the reverse of each applied gate. This technicality does not matter when defining the substitutions for self-reversible gates such as Toffoli and Fredkin gates, but must be upheld with Peres gates. If we were to implement any other non-self-reversible gate such as the Margolus gate, its substitutions must also take into account this directionality.

3.7 Synthesis Example

To provide examples using the various substitutions, we demonstrate the process of synthesizing the specification given in Eq. (5). Because the circuits shown in Figures 3, 5, and 6 can all be synthesized with the extended RMRLS (we will see later in Section 4.4 how the circuit in Figure 4 can also be obtained), an exhaustive search would actually encounter all three solutions, as shown in Figure 8. The synthesis example shown depicts various search paths that reach these solutions. A large number of other search nodes, many of which do

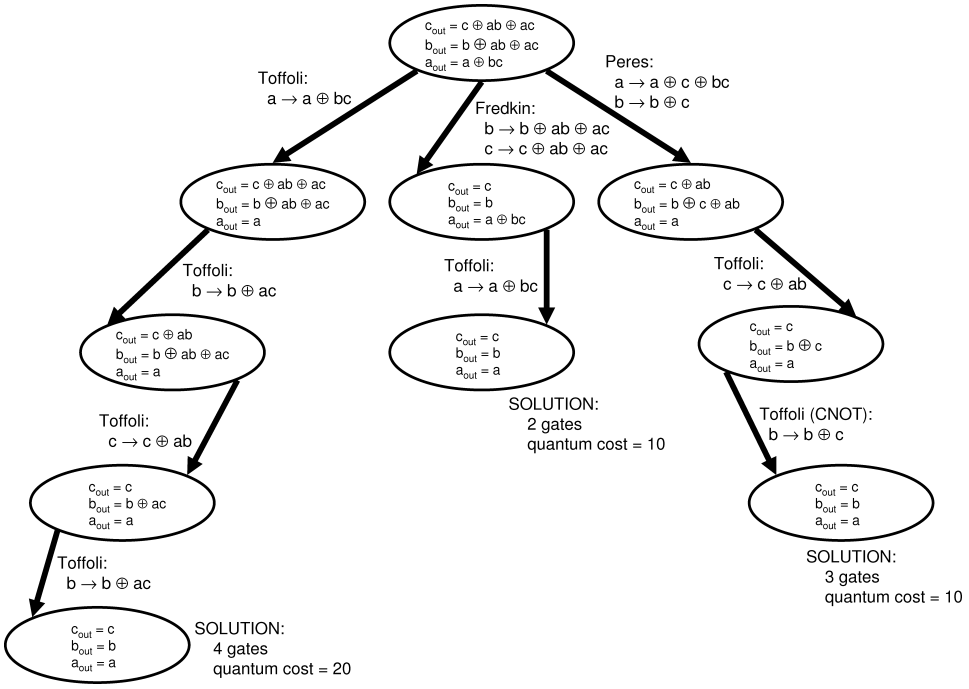

Fig. 8. Various search paths taken in the synthesis of the reversible specification given in Eq. (5).

not lead to solutions, would also be covered in an exhaustive search. For clarity, these have been omitted in the figure.

4. EXPERIMENTAL RESULTS

We next provide experimental results.

4.1 Functions of Three Variables

A standard practice for comparing heuristic reversible logic synthesis algorithms is to examine the success at gate minimization for all three-variable reversible functions. We obtain these results by running synthesis on all 40,320 possible three-variable functions. Table I shows our synthesis results in NCTS, NCTSF, and NCTSFP modes, which are formed by enabling the SWAP (S), n-bit Fredkin (F), and Peres (P) gates.

We compare our results against various other heuristic algorithms, as well as the optimal results reported by Shende et al. [2003] and Dueck et al. [2003]. Among heuristic algorithms, RMRLS provides better results than those of the extensive template libraries by Maslov et al. [2003]. Not shown in the table are the NCTS results for templates [Maslov et al. 2003], which have been shown inferior to the NCTS results by Kerntopf [2004]. Also not shown are the NCTSF results from Kerntopf [2004], which were unable to improve upon the NCTSF template library results [Maslov et al. 2003]. We have not reported the optimal gate counts for NCTSFP, as we were unable to find such data in the

Table I. Synthesis Results for All Three-Variable Reversible Functions

No. Gates	Miller et al. [2003] NCTS	Kerntopf [2004] NCTS	Maslov et al. [2003] NCTSF	Optimal [Shende et al. 2003] NCT	Optimal [Shende et al. 2003] NCTS	Optimal [Dueck et al. 2003] NCTSF	RMRLS [Gupta et al. 2006] NCT	RMRLS NCTS	RMRLS NCTSF	RMRLS NCTSFP
11	5									
10	110									
9	792	86	9				36	2		
8	4,726	2,740	512	577	32		3,351	574	18	
7	11,199	11,774	5,503	10,253	6,817	496	12,476	9,242	2,201	18
6	12,076	13,683	13,914	17,049	17,531	14,134	13,596	15,893	15,105	3,567
5	7,518	8,068	13,209	8,921	11,194	17,695	7,479	9,998	15,521	19,786
4	2,981	3,038	5,680	2,780	3,752	6,474	2,642	3,617	5,984	13,198
3	767	781	1,290	625	844	1,318	625	844	1,288	3,290
2	130	134	184	102	134	184	102	134	184	430
1	15	15	18	12	15	18	12	15	18	30
0	1	1	1	1	1	1	1	1	1	1
Avg.	6.18	6.01	5.44	5.87	5.63	5.13	6.10	5.75	5.25	4.56

RMRLS improves upon all other heuristic algorithms in both the NCTS and NCTSF categories. The average gate count also reduces significantly with the addition of Peres gates.

literature. Overall, the extended RMRLS provides better gate counts than all existing heuristic NCTS and NCTSF algorithms, and to the best of our knowledge this is the first algorithm to target the NCTSFP gate library.

4.2 Other Benchmarks

One of the significant capabilities of the RMRLS algorithm is its ability to synthesize functions of as many as 16 variables [Gupta et al. 2006], whereas some other algorithms may become impractical for such a regime. Since our extensions allow types of gates to be optionally specified by the user, at the very least the modified RMRLS retains all the capabilities of the original algorithm. The question then remains as to whether enabling new gates retains the algorithm's usability and can further reduce the gate counts for large functions.

To answer this question, we use the same benchmark suite as in Gupta et al. [2006] and synthesize all benchmarks with Fredkin and Peres gates enabled. All of these benchmarks are included in the public releases of the original RMRLS, as well as the new version featured in this work. Each benchmark was attempted in the greedy, exhaustive, and default modes, and the best result was chosen from among these. Given infinite time and memory, the exhaustive mode could theoretically always provide the best results. On our system, however, for several of the larger functions an exhaustive search cannot be completed even in 24 hours. Most results shown were obtainable in the default timeout limit of 180 seconds. The *cycle* and *adder* functions, however, have relatively large specifications and thus took much longer. Benchmarks such as these were given extra time, ranging up to 1 hour.

We compare our results in terms of gate count and quantum cost. Although RMRLS was originally designed to optimize only the gate count, improved quantum cost over results from existing algorithms has sometimes been a side-effect [Gupta et al. 2006]. Building upon this fact, we reasoned that the algorithm can focus on any metric, so we added an option to optimize for quantum cost. We report our gate counts and quantum costs in two categories in Table II. Shown are our best results when optimizing for the minimum number of gates, as well as when optimizing for reduced quantum cost.

In calculating quantum costs, we use the table and rules from Maslov [2007]. The cost of a Fredkin gate is typically obtained by taking the cost for a Toffoli gate with the same number of inputs, and then adding 2 to account for the two CNOT gates which can combine with a Toffoli gate to form a controlled swap. If all cost calculations were of this form, the best possible quantum cost would not improve with Fredkin gates. There is, however, a known implementation of three-bit Fredkin gates with a quantum cost of 5, which is the same as the cost of a three-bit Toffoli gate. When such gates are usable, the addition of Fredkin gates can serve to reduce the quantum cost. The Peres gate itself is also a special case, with a quantum cost of 4, even though an equivalent circuit using Toffoli gates would consist of a three-bit Toffoli gate alongside a CNOT gate for a total cost of 6.

As one example of a synthesized circuit, our minimum-cost realization of the *majority*5 function is as follows: $TOF3(a, b; e)$ $TOF2(a; b)$ $RPER(b; d; e)$

Table II. Synthesis Results for Reversible Logic Benchmarks in the NCTSFP Mode Compared to the Basic NCT Mode of the Original RMRLS

Benchmark	Real Inputs	Constant Inputs	Gates NCT [Gupta et al. 2006]	Cost NCT [Gupta et al. 2006]	Gates NCTSFP min-gates	Cost NCTSFP min-gates	Gates NCTSFP min-cost	Cost NCTSFP min-cost
2of5	5	2	20	100	20	96	20	9
alu	5	0	19†	163†	16	171	21	119
rd32	3	1	4	12†	4	12	5	9
rd53	5	2	13	116	13	91	17	78
3_17	3	0	6	14	5	12	5	11
4_49	4	0	13	61	10	36	12	29
xor5	5	0	4	4	4	4	4	4
4mod5	4	1	5	13	5	13	5	13
5mod5	5	1	11	91	11	93	11	91
ham3	3	0	5	9	4	9	4	7
ham7	7	0	—†	—†	20	76	22	67
hwb4	4	0	15	35	9	27	10‡	19‡
hwb5	5	0	—	—	26	196	35	175
decod24	4	0	11	31	10	48	11	30
cycle10_2	12	0	27	1,469	17	1,198	24	1,060
cycle15_2	17	0	41†	4,201†	27	3,242	27	3,242
cycle28_2	30	0	80†	19,105†	78	19,101	78	19,101
5one013	5	0	19	95	16	97	16	93
5one245	5	0	20	104	14	101	18	79
6one135	6	0	5	5	5	5	5	5
6one0246	6	0	6	6	6	6	6	6
majority3	3	0	4	16	3	14	4	13
majority5	5	0	16	104	13	94	14	81
graycode6	6	0	5	5	5	5	5	5
graycode10	10	0	9	9	9	9	9	9
graycode20	20	0	19	19	19	19	19	19
mod5adder	6	0	19	127	16	139	18	103
mod15adder	8	0	10	71	7	65	11	54
mod32adder	10	0	15	154	11	146	19	127
mod64adder	12	0	26	333	20	346	20	289

†Some results by Gupta et al. [2006] are not repeatable with the publicly available RMRLS due to modified heuristics in the released version. For these, we have compared to our best obtainable NCT results running on the released version.

‡In only one case, namely *hwb*4, did the extended branch heuristic improve the best obtainable quantum cost.

$TOF3(b,c;e)$ $TOF2(a;c)$ $TOF3(b,d;c)$ $TOF3(b,c;a)$ $TOF3(a,e;d)$ $PER(a;d;b)$ $TOF4(b,d,e;a)$ $PER(c;b;a)$ $FRED4(d,e;a,b)$ $TOF2(c;a)$ $TOF4(a,d,e;c)$ for a quantum cost of 81 realized with 14 gates. We denote Toffoli gates as $TOF\#(cbit_1 \ldots cbit_{n-1};tbit)$, where # denotes the size of the gate, the $cbit$ terms designate zero or more control bits, and $tbit$ designates the target bit. Fredkin gates have a similar notation of $FRED\#(cbit_1 \ldots cbit_{n-2};tbit_1,tbit_2)$ where there are two target bits. Peres gates are denoted as $PER(cbit;tbit_{TOF};tbit_{CNOT})$, where $tbit_{TOF}$ signifies the target bit for the output characteristic of a three-bit Toffoli gate and $tbit_{CNOT}$ represents the remaining target bit. Reverse-Peres gates are denoted by *RPER* with the same format.

Overall, our results show that the addition of Fredkin and Peres gates can always match, and often improve, the gate count of various benchmarks. When optimizing for quantum cost, we often obtain different circuits possibly with an increased gate count, but always with a quantum cost that matches or

improves upon the circuits obtained by Gupta et al. [2006] or by the heuristic that optimizes gate count. Furthermore, the addition of Fredkin and Peres gates allowed the synthesis of *ham*7 and *hwb*5, which (within the constraints of our 4GB RAM and noninfinite runtime) are not synthesizable in NCT-only mode.

4.3 Runtime

According to Gupta et al. [2006], RMRLS was shown extremely robust in terms of its ability to find solutions within a practical amount of time. Our extensions for additional gates can be turned off at runtime. Thus, if the goal is to find a solution, even if it is suboptimal, then at the very least a user can obtain this solution using the NCT mode only.

As expected, using additional gates in the search tree does slow down exhaustive searches. This is because the additional branches at each node can further spawn more branches, effectively increasing the exponential growth rate of the search tree. In many cases, we found that exhaustive searches, when including Fredkin and Peres gates, could take up to four times as long as their corresponding NCT runtimes.

On the other hand, we did not see the same increase in runtime for greedy searches where the purpose was to find any solution. In fact, when asked to find any (i.e., not necessarily optimal) solution, for the majority of benchmarks in Table II, we were able to obtain some solution in less than a second. The only benchmarks that took longer than 10 seconds to obtain a solution in the greedy mode were *ham*7 (67 seconds), *hwb*5 (77 seconds), and 5*one*245 (39 seconds).

All results in this article were obtained on a platform utilizing an AMD Athlon X2 2.0 GHz processor along with 4GB of RAM running Fedora Core Linux.

4.4 Extended Branches Heuristic

The discrepancy between the circuit given in Figure 4 and the equivalent circuit in Figure 5 raises an issue for the original substitution rules of RMRLS. Even in exhaustive search mode, RMRLS would never come across the solution shown in Figure 4, simply because all of the candidate factors encountered contain at least two variables and thus result in transformations representing only three-bit Toffoli gates. In order to obtain the circuit in Figure 5, which has a reduced quantum cost, the algorithm would have to traverse branches signifying CNOT gates, even when a one-variable candidate factor is not available.

An even simpler case that signifies this problem can be found when attempting to synthesize a three-bit Fredkin gate using only Toffoli gates. As revealed in Gupta et al. [2006], RMRLS cannot synthesize the minimum quantum cost solution for this basic case, even if its solution is minimal in terms of the gate count of 3. In general, the observed effect is that lacking any protection against these cases can result in circuits with higher-than-expected quantum cost. However, we would not necessarily expect much difference in gate count.

To address this problem, we provide the option of additional substitutions. Like conventional substitutions, these are still based on detected candidate

factors. Our special rule, however, is that for any given *factor* as depicted in Eq. (6), we may substitute a term based on a subset of the variables in *factor*. For instance, if a substitution of the form $a \rightarrow a \oplus bc$ is suggested by a candidate factor, this substitution would be performed in addition to $a \rightarrow a \oplus b$, $a \rightarrow a \oplus c$, and $a \rightarrow a \oplus 1$. The same expansion is also applied to Fredkin candidates. With this extended branching heuristic, there can quickly arise an intractable number of nodes in the search tree. Thus, at the current time this heuristic is only usable for functions of relatively few variables.

Among all benchmarks, the only one upon which this heuristic improved was *hwb*4, with its quantum cost reduced from 27 to 19.

4.5 n-Bit Peres Gates

The bulk of this article assumes three-bit Peres gates. We chose this definition because it is the accepted one used by the reversible logic community, and assuming a system of n-bit Peres gates would exaggerate the gate count reduction for our benchmark results. Nonetheless, it is interesting to consider how our results would change if we assumed n-bit Peres gates, as well as n-bit reverse-Peres gates.

In general, any n-bit Peres gate serves as the equivalent of an n-bit Toffoli gate followed by an $(n-1)$-bit Toffoli gate, as long as the control bits and target bit of the smaller Toffoli gate form a subset of the control bits of the larger gate. A reverse-Peres gate performs a similar function, except that the equivalent circuit would have the smaller Toffoli gate placed first.

Because there are no known quantum costs for Peres gates that are better than their equivalent Toffoli-gate-based realizations, it cannot be assumed that the use of n-bit Peres gates will reduce the quantum cost, at least under our current cost calculation schemes. However, n-bit Peres gates would be expected to reduce the gate counts of various reversible benchmarks. If it is later found that n-bit Peres gates also have quantum cost less than their corresponding n-bit Toffoli gates (as in the three-bit case) it will make sense to expect n-bit Peres gates to be a fundamental building block in gate libraries.

Table III shows the subset of benchmarks that can be synthesized with fewer gates using the n-bit Peres gate assumption. Only 10 out of the original 30 benchmarks were able to utilize Peres gates of more than 3 bits. Regarding quantum cost, with only a pessimistic assumption for higher-ordered Peres gates, these realizations do not reduce the cost under our calculation methods. Thus, we have only listed the number of gates.

As a Peres gate fulfills the function of two Toffoli gates in sequence, the *cycle* benchmarks benefit by far the most from the n-bit Peres gate assumption. Realizations of *cycle* functions typically include long chains of Toffoli gates operating on a large set of control bits gradually decreasing in size. Replacing each pair of Toffoli gates with a single Peres gate effectively halves the total number of gates.

While in an ideal situation, the n-bit Peres gate assumption could reduce circuit size by half, this does not explain how the circuit size of *cycle*28_2 reduced by even more than half. Taking this realization into account and decomposing

Table III. Gate Counts for the 10 Benchmarks From Table II That Can Be Synthesized With Fewer Gates if the NCTSFP Library Assumes Peres Gates of More Than Three Inputs.

Benchmark	Gates NCTSFP Three-bit Peres	Gates NCTSFP *n*-bit Peres
alu	16	14
ham7	21	18
cycle10_2	17	12
cycle15_2	27	16
cycle28_2	78	32
5one013	16	15
majority5	13	11
mod15adder	7	6
mod32adder	11	9
mod64adder	20	18

the larger Peres gates back into their Toffoli equivalents results in a gate count of 59, which is better than the other realization of 78 gates. This appears to be a result of the unpredictable side-effects of heuristic pruning under different configurations.

5. CONCLUSIONS

This work builds upon the state-of-the-art reversible logic synthesis algorithm to include Fredkin and Peres gates. We have shown that the process of adding new gates to the algorithm is a generalizable methodology applicable to any arbitrary gate.

For three-variable functions we show that, when including SWAP and Fredkin gates, RMRLS outperforms all other known heuristic methods. Including Peres gates as well reduces the average gate count for three-variable functions to 4.56, notably better than even the optimal case of 5.13 when using Toffoli, SWAP, and *n*-bit Fredkin gates. For reversible benchmarks, the additional gate types are often able to reduce circuit size in terms of either gate count or quantum cost, depending on the option requested by the user.

We have implemented these extensions into RMRLS such that the combination of gate types, limitations on these gates, and the choice of whether to optimize for gate count or quantum cost can be chosen by the user at runtime. This new release of RMRLS is freely available for download at http://www.princeton.edu/∼cad/.

ACKNOWLEDGMENTS

We would like to thank Pallav Gupta for his assistance with RMRLS, and the anonymous reviewers for their helpful comments.

REFERENCES

AGRAWAL, A. AND JHA, N. K. 2004. Synthesis of reversible logic. In *Proceedings of the Conference and Exhibition on Design, Automation and Test in Europe (DATE)*, vol. 2, 1384–1385.

BARENCO, A., BENNETT, C. H., CLEVE, R., DIVINCENZO, D. P., MARGOLUS, N., SHOR, P., SLEATOR, T., SMOLIN, J., AND WEINFURTER, H. 1995. Elementary gates for quantum computation. *Phys. Rev. A. 52*, 3457–3467.

BENNETT, C. H. 1973. Logical reversibility of computation. *IBM J. Res. Dev. 17*, 6, 525–532.

CUYKENDALL, R. AND ANDERSEN, D. R. 1987. Reversible optical computing circuits. *Optics Lett. 12,* 7, 542–544.

DE VOS, A. 1994. Proposal for an implementation of reversible gates in CMOS. *Int. J. Electron. 76*, 293–302.

DUECK, G. W., MASLOV, D., AND MILLER, D. M. 2003. Transformation-based synthesis of networks of Toffoli/Fredkin gates. In *Proceedings of the IEEE Canadian Conference on Electrical and Computer Engineering,* 211–214.

FREDKIN, E. AND TOFFOLI, T. 1982. Conservative logic. *J. Theor. Phys. 21,* 219–253.

GUPTA, P., AGRAWAL, A., AND JHA, N. K. 2006. An algorithm for synthesis of reversible logic circuits. *IEEE Trans. Comput. Aided Des. Integr. Circ. Syst. 25*, 11(Nov.), 2317–2330 (tool available for download: http://www.princeton.edu/~cad/).

HUNG, W. N. N., SONG, X., YANG, G., YANG, J., AND PERKOWSKI, M. 2004. Quantum logic synthesis by symbolic reachability analysis. In *Proceedings of the Conference and Exhibition on Design, Automation and Test in Europe (DATE)*, 838–841.

IWAMA, K., KAMBAYASHI, Y., AND KAMASHITA, S. 2002. Transformation rules for designing CNOT-based quantum circuits. In *Proceedings of the Conference and Exhibition on Design, Automation and Test in Europe (DATE)*, 419–424.

KERNTOPF, P. 2004. A new heuristic algorithm for reversible logic synthesis. In *Proceedings of the Conference and Exhibition on Design, Automation and Test in Europe (DATE)*, 834–837.

LANDAUER, R. 1961. Irreversibility and heat generation in the computing process. *IBM J. Res. Dev. 5* (Jul.), 183–191.

MARGOLUS, N. 1988. Physics and computation. Ph.D. dissertation, Massachusetts Institute of Technology, Cambridge, MA.

MASLOV, D. 2003. Reversible logic synthesis. Ph.D. dissertation, The University of New Brunswick, Fredericton, New Brunswick, Canada.

MASLOV, D. AND DUECK, G. W. 2004. Reversible cascades with minimal garbage. *IEEE Trans. Comput. Aided. Des. Integr. Circ. Syst. 23*, 11 (Nov.), 1497–1509.

MASLOV, D., DUECK, G. W., AND SCOTT, N. 2007. Reversible logic synthesis benchmarks page. http://www.cs.uvic.ca/~dmaslov/.

MASLOV, D., DUECK, G. W., AND MILLER, D. M. 2005. Toffoli network synthesis with templates. *IEEE Trans. Comput. Aided Des. Integr. Circ. Syst. 24,* 6 (Jun.), 807–817.

MASLOV, D., DUECK, G. W., AND MILLER, D. M. 2003. Fredkin/Toffoli templates for reversible logic synthesis. In *Proceedings of the IEEE-ACM International Conference on on Computer-Aided Design,* 256–261.

MILLER, D. M. 2002. Spectral and two-place decomposition techniques in reversible logic. In *Proceedings of the IEEE Midwest Symposium on Circuits and Systems,* vol. 2,493–496.

MILLER, D. M., MASLOV, D., AND DUECK, G. W. 2003. A transformation-based algorithm for reversible logic synthesis. In *Proceedings of the ACM-IEEE Design Automation Conference (DAC),* 318–323.

PERES, A. 1985. Reversible logic and quantum computers. *Phys. Rev. A 32*, 3266–3276.

PERKOWSKI, M., KERNTOPF, P., BULLER, A., CHRZANOWSKA-JESKE, M., MISHCHENKO, A., SONG, X., AL-RABADI, A., JOZWIAK, L., COPPOLA, A., AND MASSEY, B. 2001. Regular realization of symmetric functions using reversible logic. In *Proceedings of the EUROMICRO Symposium on Digital Systems Design,* 245–252.

SHENDE, V. V., PRASAD, A. K., MARKOV, I. L., AND HAYES, J. P. 2003. Synthesis of reversible logic circuits. *IEEE Trans. Comput. Aided Des. Integr. Circ. Syst. 22*, 6 (Jun.), 710–722.

SMOLIN, J. A. AND DIVINCENZO, D. P. 1996. Five two-bit quantum gates are sufficient to implement the quantum Fredkin gate. *Phys. Rev. A 53*, 2855–2856.

TOFFOLI, T. 1980. Reversible computing. In *Automata, Languages and Programming*, J. W. de Bakker and J. van Leeuwen, eds. Springer, 632–644.

Received March 2007; revised July 2007; accepted October 2007 by Frederic Chong