A Transformation Based Algorithm for Reversible Logic Synthesis

D. Michael Miller
Dept. of Computer Science
University of Victoria
Victoria, BC, V8W 3P6
Canada
mmiller@csr.uvic.ca

Dmitri Maslov
Faculty of Computer Science
University of New Brunswick
Fredericton, NB, E3B 5A3
Canada
dmaslov@unb.ca

Gerhard W. Dueck
Faculty of Computer Science
University of New Brunswick
Fredericton, NB, E3B 5A3
Canada
gdueck@unb.ca

ABSTRACT

A digital combinational logic circuit is reversible if it maps each input pattern to a unique output pattern. Such circuits are of interest in quantum computing, optical computing, nanotechnology and low-power CMOS design. Synthesis approaches are not well developed for reversible circuits even for small numbers of inputs and outputs.

In this paper, a transformation based algorithm for the synthesis of such a reversible circuit in terms of $n \times n$ Toffoli gates is presented. Initially, a circuit is constructed by a single pass through the specification with minimal look-ahead and no back-tracking. Reduction rules are then applied by simple template matching. The method produces near-optimal results for 3-input circuits and also produces very good results for larger problems.

Categories and Subject Descriptors

M1.8 [**Design Methodologies**]: Logic Design

General Terms

Design, Theory

Keywords

Reversible Logic, Quantum Circuits, Templates, Minimization

1. INTRODUCTION

Landauer [8] proved that using traditional irreversible logic gates necessarily leads to power dissipation regardless of the underlying technology. Further, Bennett [1] showed that for power not to be dissipated in an arbitrary circuit, it must be built from reversible gates. Hence there are compelling reasons to consider circuits composed of reversible gates. Reversible circuits are of particular interest in low-power CMOS design [17], optical computing [7], quantum computing [14], and nanotechnology [10].

An $n \times n$ Toffoli gate [19] has $n-1$ control lines which pass through the gate unaltered and a target line on which the value is inverted if all the control lines have value '1'. In this paper, we present a fast synthesis algorithm which accepts a reversible function specification and produces a reversible circuit composed of $n \times n$ Toffoli gates [19].

The synthesis of reversible circuits differs significantly from synthesis using traditional irreversible gates. Approaches have been presented in [5, 6, 11, 13, 16, 18]. For many of those methods extensive searching is required. A key factor in this contribution is that we avoid extensive searching and therefore the method has greater potential to be extended to functions with more than just a few inputs and outputs.

We first give a basic naive algorithm which synthesizes the circuit in one direction. We show that this algorithm will always complete without introducing unnecessary garbage outputs and that the circuit will have at most $(m-1)2^m+1$ gates. Output permutation and an heuristic for minimizing gate width are then introduced.

Next we show that the approach can be applied in both directions simultaneously with gates being identified at either the input or the output end of the circuit whichever offers best advantage as the synthesis proceeds. Transformations to reduce the number of gates are applied using template matching. Our set of transformations is an expansion of the those used in [18] and [5].

Necessary background is reviewed in Section 2. Our synthesis approach is described in Section 3 and gate transformation by template matching is discussed in Section 4. Experimental results are presented in Section 5. By considering all $8! \ 3 \times 3$ reversible functions we show that our algorithm produces results quite close to the optimal circuit sizes found by exhaustive search in [18]. We also demonstrate by examples that our method can be applied to larger functions and to the realization of irreversible functions. Section 6 concludes the paper with observations and suggestions for further research.

2. BACKGROUND

DEFINITION 1. *An m-input, m-output, totally-specified Boolean function $f(X)$, $X = \{x_1, x_2, ..., x_m\}$ is reversible if it maps each input assignment to a unique output assignment.*

DAC 2003, June 2–6, 2003, Anaheim, California, USA.

c	b	a	c'	b'	a'
0	0	0	1	1	1
0	0	1	0	0	1
0	1	0	1	0	0
0	1	1	0	1	1
1	0	0	0	0	0
1	0	1	0	1	0
1	1	0	1	1	0
1	1	1	1	0	1

Table 1: 3×3 Reversible Logic Function.

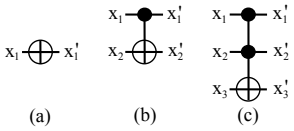

Figure 1: (a) $TOF1(x_1)$, (b) $TOF2(x_1, x_2)$ and (c) $TOF3(x_1, x_2, x_3)$ Toffoli Gates.

A reversible function can be written as a standard truth table as in Table 1 and can also be viewed as a bijective mapping of the set of integers $0, 1, ..., 2^m - 1$ onto itself. Hence a reversible function can be defined as an ordered set of integers corresponding to the right side of the table, *e.g.* $\{7, 1, 4, 3, 0, 2, 6, 5\}$ for the function in Table 1. We can thus interpret the function over the integers as $f(0) = 7, f(1) = 1, f(2) = 4$, *etc.*

A reversible function is, of course, a permutation and can be expressed as a set of disjoint cycles as done in [18], but we do not follow that approach here.

DEFINITION 2. *An n-input, n-output gate is reversible if it realizes a reversible function.*

A variety of reversible gates have been proposed [19, 3, 4]. Here we use the family of Toffoli gates [19] defined as follows:

DEFINITION 3. *An $n \times n$ Toffoli gate passes the first $n-1$ lines (control) through unchanged, and inverts the n^{th} line (target) if the control lines are all 1.*

We shall write an $n \times n$ Toffoli gate as $TOFn(x_1, x_2, ..., x_n)$ where x_n is the target line. Using the prime symbol to denote the value of a line after passing through the gate we have

$$x_i' = x_i, \quad i < n, \tag{1}$$

$$x_n' = x_1 x_2 ... x_{n-1} \oplus x_n \tag{2}$$

$TOF1(x_1)$ is the special case where there are no control inputs, so x_1 is always inverted, *i.e.* it is a NOT gate. $TOF2(x_1, x_2)$ has been termed a Feynman [3] or controlled-NOT gate (CNOT). $TOF3(x_1, x_2, x_3)$ is often referred to simply as a Toffoli gate [19]. These gates are depicted as shown in Figure 1.

DEFINITION 4. *A SWAP gate exchanges a pair of inputs.*

DEFINITION 5. *Given two bit strings, p and q, the Hamming distance between them, denoted $\delta(p, q)$ is the number of positions for which p and q differ.*

DEFINITION 6. *Given the function $f(X)$, the complexity $C(f)$ is defined as the the sum of the individual Hamming distances over the 2^m input-output patterns.*

For example, the value of $C(f)$ for the function in Table 1 is 10.

3. THE ALGORITHM

Applying a Toffoli gate to the inputs or the outputs of a reversible function always yields a reversible function. The synthesis problem is to find a sequence of Toffoli gates which transforms a given reversible function to the identity function. As gates can be applied either to the inputs or the outputs, the synthesis can proceed from outputs to inputs, inputs to outputs or, as we show in Section 3.3, in both directions simultaneously.

3.1 Basic Algorithm

To begin, we present a basic naive and greedy algorithm which identifies Toffoli gates only on the output side of the specification.

Consider, a reversible function specified as a mapping over $\{0, 1, ..., 2^m - 1\}$.

Basic Algorithm

Step 1: If $f(0) \neq 0$, invert the outputs corresponding to 1-bits in $f(0)$. Each inversion requires a $TOF1$ gate. The transformed function f^+ has $f^+(0) = 0$.

Step 2: Consider each i in turn for $1 \leq i < 2^m - 1$ 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 with $f^{++}(i) = i$. The required gates must map $f^+(i) \rightarrow i$.

Let p be the bit string with 1's 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 1's in all positions where the expansion of i is 0 while the expansion of $f^+(i)$ is 1. q identifies the 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. Then, for each $q_k = 1$, apply the Toffoli gate with control lines corresponding to all outputs in positions where the expansion of $f^+(i)$ is 1 and whose target line is the output in position k.

For each $1 \leq i < 2^m - 1$, Step 2 transforms $f^+(i) \rightarrow i$ by applying the specified sequence of Toffoli gates. Since we consider the i in order, and step 1 handles the case for 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 Step 2 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 $2^m - 1$ values.

	(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**	1**0**1	101
110	100	10**1**	101	1**1**1	11**0**
111	110	11**1**	11**0**	110	11**1**

Table 2: Example of applying the basic algorithm.

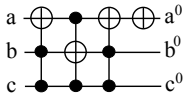

Figure 2: Circuit for the function shown in Table 2.

Table 2 illustrates the application of the basic algorithm. (i) is the given specification. Step 1 identifies the application of $TOF1(a^0)$ giving (ii). At this point $f^+(i), 0 \leq i \leq 4$ are as required. Mapping $f^+(5) \rightarrow 5$ requires $TOF3(c^1, b^1, a^1)$ to change the rightmost position to 1 (iii) and $TOF3(c^2, a^2, b^2)$ to remove the centre 1 (iv). Lastly, $TOF3(c^3, b^3, a^3)$ 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 corresponding circuit is shown in Figure 2.

The basic algorithm is straightforward and easily implemented. Its algorithmic complexity is $n2^n$. It is also easily seen that it will always terminate successfully with a circuit for the given specification. However, it is possible to construct a function for any m, that requires $(m-1)2^m + 1$ gates. For $m = 3$, this is the function shown in Table 1. We next consider a number of approaches to reduce the size of the circuit produced.

3.2 Output Permutation and Control Input Reduction

The basic algorithm maps each output back to the corresponding input. Often this is not the best mapping. For functions with up to 8 or 9 inputs it is practical to try all $m!$ output permutations. Permuting the outputs requires a certain number of interchanges which in some technologies may require explicit SWAP gates.

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 a row earlier in the specification. This is easily accounted for since the set of control lines must either contain a line that has not appeared as a 1 in an 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. Recall, that the complexity C is the total Hamming distance between the input and output sides of the specification, so this heuristic is choosing the gate that moves the specification furthest towards the identity specification. In case of a tie, the smallest set of control lines is used, and within that the choice is arbitrary.

	(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

Table 3: Example of applying the bidirectional algorithm.

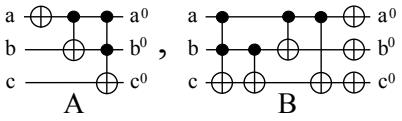

Figure 3: Circuit for the function shown in Table 3.

3.3 Bidirectional Algorithm

As described so far, the algorithm produces the circuit by selecting Toffoli gates manipulating only the output side of the specification. Since the specification is reversible, one could consider the inverse specification deriving a reverse circuit and then choose whichever is the smaller. A better approach is to apply the method in both directions simultaneously choosing to add gates at the input side or the output side.

To see how this works, consider the initial reversible specification in Table 3, 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 $TOF1(a)$ 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 $TOF2(a, b)$. 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 $TOF3(a, b, c)$ (when expressed in terms of the input lines) and the circuit is done (iv). The result uses three gates (shown in Figure 3 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 3 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, $j > i$ and must always exist. Also, the same rules for identifying the control lines, including reduction, described above apply. Our bidirectional algorithm chooses (a) if $\delta(i, f^+(i)) \leq \delta(i, j)$, and (b) otherwise. 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.

4. TEMPLATE MATCHING

The circuits produced by the algorithm as described thus far frequently have gate sequences that can be reduced. For example, the sequence $TOF2(b,a), TOF1(b), TOF1(a)$ can be replaced by the sequence $TOF1(b), TOF2(b,a)$. We have implemented a template driven reduction method. A template consists of a sequence of gates to be matched and the sequence of gates to be substituted when a match is found. The lines in the template are generic and must be associated to real lines in the circuit with the association applied consistently across the template. This is accomplished by first associating the widest target template gate with a gate in the circuit and then searching the circuit for the other target gates using the line association derived from the widest gate. Note that since the order of the control lines to a Toffoli gate is immaterial, $c!$ line associations must be considered where c is the number of control lines for the widest gate.

Our template matching procedure looks for the target gates, including the initial match to the widest gate, across the entire circuit. If all target gates are found, it attempts to move the gates so that they are adjacent either matching the template in the forward or reverse direction. If this can be done, the matched gates are replaced with the new gates specified by the template. For a reverse match, the new gates are substituted in reverse order.

When moving the target gates, the matching procedure takes account of Property 1 which follows directly from the definition of $n \times n$ Toffoli gates. If two gates can not be interchanged because they don't satisfy this property and that prohibits proper adjacent ordering of the target gates for a match, the template being considered is not applicable.

PROPERTY 1. *Two gates* $TOF_k(x_1, x_2, ..., x_{k-1}, x_k)$ *and* $TOF_l(y_1, y_2, ..., y_{l-1}, y_l)$ *adjacent in a circuit can be interchanged iff* $x_k \notin \{y_1, y_2, ..., y_{l-1}\}$ *and* $y_l \notin \{x_1, x_2, ..., x_{k-1}\}$.

Our matching procedure tries all appropriate sets of target gates for each template. When a template match is found, the substitution dictated by the template and the process restarts since a substitution may mean that a template rejected earlier becomes applicable.

Figure 4 shows the current template set employed by our procedure. Templates 2.1, 3.1 - 3.3, and 4.1 - 4.3 were introduced in [5]. We have classified the templates as follows (classes are separated by horizontal lines):

(1) two inputs involving SWAPs;

(2) two input gate reductions without SWAPs;

(3) transformation rule 3 from [5];

(4) symmetric templates;

(5) controlled SWAP (equivalent to the Fredkin gate).

It can be shown that a generalization of classes (3) and (4) generates all templates with n inputs and 3 gates that result in a reduction in the number of gates.

5. EXPERIMENTAL RESULTS

Table 5 shows the results of applying various versions of our algorithm to all $8! = 40320$ 3×3 reversible functions. The four scenarios are:

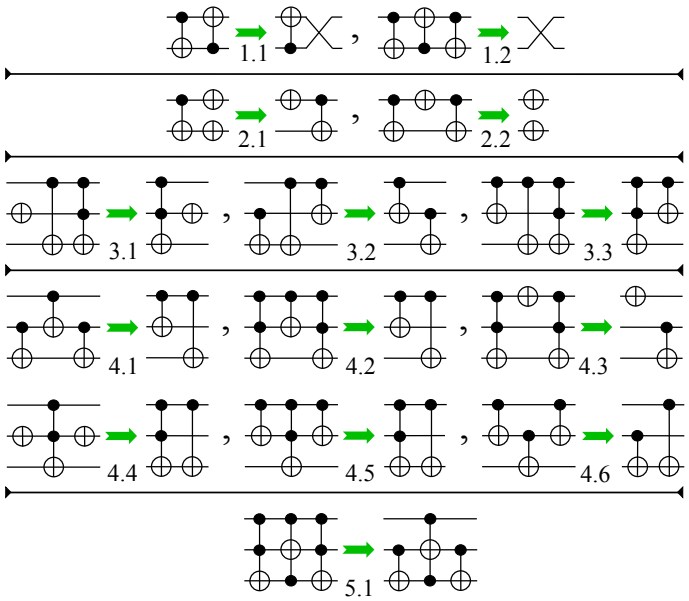

Figure 4: Templates with 2 or 3 inputs.

(a) the basic output transformations algorithms;

(b) (a) plus Hamming distance based look-ahead;

(c) (b) plus bidirectional transformation;

(d) (c) plus template application.

For each scenario, we show the number of functions for each gate count, the average number of gates required, and the total time to apply our method for the 8! functions on a PC with a 750MHz Pentium III with 256 Mb RAM.

Column (e) in Table 5 shows the optimal results reported in [18]. That work used depth-first search with iterative deepening to construct optimal gate count circuits for $n = 3$. However, this approach does not scale-up to larger functions. For example, while the optimal results for $n = 3$ were found in 15 sec. using a PC with a 2 GHz Pentium-4 Xeon, the authors report that a 4×4 reversible function requiring 8 or less gates can be synthesized in less than a second whereas the synthesis requires more than 1.5 hours when 9 or more gates are required. As we will show below by example, our approach is applicable to larger functions in reasonable time.

Table 5 compares the advantages of the various refinements to our method. The full bidirectional algorithm with output permutation, control input reduction and template matching produces results quite comparable to the optimal results. The table does not indicate the true advantage of control input reduction. Overall, the average gate count is essentially the same as without this refinement, but the gates require fewer inputs for many circuits. Alternative heuristics for reducing the gate input count need to be considered.

An irreversible function can be realized using reversible gates [14]. Garbage outputs must be added as 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 extending the irreversible function specification to a larger reversible one.

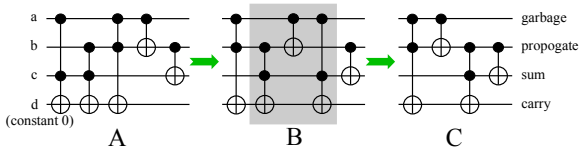

Figure 5: Full adder.

DEFINITION 7. *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 [9], 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 using the approach described in [12]. Often they can simply be set equal to input variables. At other times, we use XOR functions involving subsets of the inputs.

Constant inputs when required are defined so that the circuit yields the required functionality when they are set to 0. At present, our approach does not handle dont-cares so the reversible specification derived from the irreversible specification must be totally-specified. This is most easily accomplished by ensuring the output patterns are unique for the section of the specification for all constant inputs 0, and then completing the specification by replacing certain outputs with the XOR of the output and one of the constant inputs. Often, after an initial irreversible specification is constructed and a circuit found by applying our algorithm, it is apparent from the circuit how to replace certain of the garbage outputs with alternative definitions so that some gates in the circuit will be unnecessary. Also, all SWAP gates generated amongst garbage outputs are unnecessary and all gates which simply complete the realization of a garbage output (their target is not used as control for a gate required to realize one of the "real" outputs) can be discarded. Both these reductions of course redefine the garbage specification.

First we consider a 3-input full adder which generates, sum, carry and propagate as used in [2]. One garbage output is required since the maximum output pattern multiplicity of the full adder is 2. The garbage output is set to an input (from the symmetry of the adder it does not matter which). A single constant input (d) is required. The complete reversible specification is

$$\begin{aligned} d' &= d \oplus \text{carry}(c, b, a) \\ c' &= \text{sum}(c, b, a) \\ b' &= b \oplus a \\ a' &= a \end{aligned}$$

Our algorithm finds the 5 gate realization shown in Figure 5 (a) in 0.07 seconds. We compare this to previously obtained results. A realization with 5 Fredkin gates is shown in [2]. The complexity of the Fredkin gate is similar to the 3×3 Toffoli gate [15]. However our realization has a single garbage output, whereas the circuit in [2] has 3. This is a significant advantage if the circuit is implemented using quantum gates.

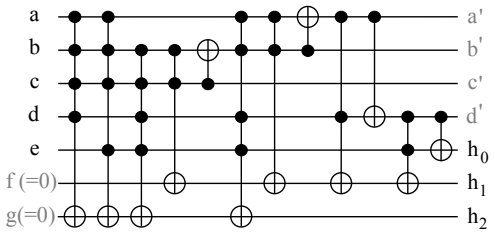

Figure 6: Circuit for $rd53$.

We now show that our adder circuit can be optimized. Note that the gates in Figure 5 (a) can be rearranged by using Property 1 as shown in (b). The three shaded gates in Figure 5 (b) are a generalization of template 3.2. When it is replaced, we have a circuit with 4 gates shown in (c). Since we have not yet implemented the generalization of templates with $n > 3$, our program was not able to find this result. This limitation is due to the set of templates currently used, and is not a limitation of the template matching approach. Our final result has the same gates as the adder obtained in [6].

As a final example we consider the benchmark function $rd53$. This function has 5 inputs and 3 outputs. The outputs are the binary encoding of the weight of the input pattern i.e. the number of 1's in the input pattern. For example, input 00000 yields output 000, input 00100 yields output 001 and input 11111 yields output 101. The maximum output pattern multiplicity is 10 so at least 4 garbage outputs must be added giving a total of at least 7 outputs. That in turn requires two inputs be added. Initially the reversible specification given in [12] was used. The garbage outputs were subsequently modified to remove unnecessary gates. Our algorithm produces a circuit with 12 gates in 1.84 seconds of CPU time. (A SWAP produced by the program was removed, since the function is symmetric.) This is better than the circuit with 14 gates proposed in [13]. Surprisingly this result is obtained when using templates with a maximum input of 3. It will be interesting to see if templates with more inputs can further reduce this circuit.

6. CONCLUSIONS

A simple algorithm for the synthesis of a reversible circuit composed of generalized Toffoli gates has been presented. The basic algorithm will always terminate with a valid circuit. Heuristic approaches have been given to reduce the size of the circuits produced through output permutation and 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 circuit synthesized in both directions simultaneously. An exhaustive examination for $m = 3$ has shown our approach yields results quite comparable to the optimal gate counts. Examples were given to show our approach can be applied to larger functions.

The fact that we currently consider all output permutations limits our methods to problemswith 8 or 9 inputs and outputs. We are studying methods for selecting a "good"

Size	(a)	(b)	(c)	(d)	(e)	(f)
17	1					
16	14					
15	92					
14	380	4	5			
13	1113	72	87			
12	2468	477	550	3		
11	4311	1759	1901	86	5	
10	6083	4179	4267	493	110	
9	7044	6912	6828	2312	792	
8	6754	8389	8221	6944	4726	12
7	5379	7766	7670	11206	11199	6817
6	3549	5615	5610	10169	12076	17531
5	1922	3183	3204	5945	7518	11194
4	839	1391	1402	2375	2981	3752
3	286	453	455	650	767	844
2	72	104	104	121	130	134
1	12	15	15	15	15	15
0	1	1	1	1	1	1
avg. gates	8.67	7.65	7.67	6.53	6.18	5.63
Time (sec.)	0.94	2.19	3.87	3.92	20.2	

(a): naive algorithm
(b): (a) plus output permutation
(c): (b) plus control input reduction
(d): (c) bidirectional reduction
(e): (d) plus template application
(f): optimal sizes [18]

Table 4: Number of reversible functions using a specified number of gates for $m = 3$.

permutation based on the initial function specification. We are also considering extensions to our method to allow don't-care conditions. Coupled with that, we are looking at ways of dynamically assigning the garbage outputs required for irreversible specifications rather than the pre-assignment method we currently use.

We are studying the extension of templates to $n > 3$. In particular, we are looking at classes of templates and generic definitions of those classes to avoid the template specification set becoming overly large and degrading the performance of the approach.

Finally, we are looking at ways to directly incorporate Fredkin gates [4] which have similar cost to Toffoli gates in some technologies, but different expressive power. A Fredkin gate is, in fact, a controlled-swap and has the same relation to a simple SWAP gate that a Toffoli gate has to a simple NOT gate. Hence, a method to incorporate Fredkin gates will also allow us to make better use of SWAP gates throughout the synthesis process.

7. REFERENCES

[1] C. Bennett. Logical reversibility of computation. *I.B.M. J. Res. Dev.*, 17:525–532, 1973.

[2] J. W. Bruce, M. A. Thornton, L. Shivakumaraiah, P. S. Kokate, and X. Li. Effiient adder circuits based on a conservative reversible logic gate. In *IEEE Symposium on VLSI*, pages 83–88, April 2002.

[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 *Proceedings of the Design Automation Conference*, New Orleans, Louisiana, USA, June 10-14 2002.

[6] A. Khlopotine, M. Perkowski, and P. Kerntopf. Reversible logic synthesis by iterative compositions. *International Workshop on Logic Synthesis*, 2002.

[7] E. Knill, R. Laflamme, and G. J. Milburn. A scheme for efficient quantum computation with linear optics. *Nature*, pages 46–52, Jan. 2001.

[8] R. Landauer. Irreversibility and heat generation in the computing process. *IBM J. Res.*, 5:183–191, 1961.

[9] 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.

[10] R. C. Merkle. Two types of mechanical reversible logic. *Nanotechnology*, 4:114–131, 1993.

[11] D. M. Miller. Spectral and two-place decomposition techniques in reversible logic. In *Midwest Symposium on Circuits and Systems*, Aug. 2002.

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

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

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

[15] M. Perkowski and et al. A hierarchical approach to computer-aided design of quantum circuits. *Preprint*, 2002.

[16] M. Perkowski, P. Kerntopf, A. Buller, M. Chrzanowska-Jeske, A. Mishchenko, X. Song, A. Al-Rabadi, L. Joswiak, A. Coppola, and B. Massey. Regularity and symmetry as a base for efficient realization of reversible logic circuits. In *International Workshop on Logic Synthesis*, 2001.

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

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

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