TABLE II
VARIABILITY RESULTS FOR DUAL-THRESHOLD VOLTAGE ALLOCATION

Benchmark	Algorithm	Delay				Leakage			
	Type	Low	Nominal	High	ΔD	Low	Nominal	High	ΔL
C1355	Nominal	4358.85	4690.00	5076.97	8.25%	406.27	1985.08	9593.97	9177.7
	Variability	4309.08	4619.18	4971.07	6.00%	606.06	2760.15	12670.20	12064.14
	Variability	4260.19	4550.29	4877.27	4.00%	871.36	3717.97	16001.40	15130.04
	Variability	4210.65	4480.94	4783.51	2.00%	1316.89	5277.11	21309.59	19992.70
C1908	Nominal	6082.91	6548.00	7094.86	8.35%	170.58	886.05	4709.74	4539.16
	Variability	6013.28	6448.94	6951.37	6.00%	251.03	1219.78	6125.34	5874.31
	Variability	5944.37	6351.69	6811.64	4.00%	373.69	1708.84	8062.22	7688.53
	Variability	5864.54	6240.16	6684.57	2.00%	555.56	2322.19	10080.26	9524.7
C2670	Nominal	8707.25	9397.00	10194.37	8.48%	388.15	1885.21	9446.47	9058.32
	Variability	8618.41	9238.11	9969.88	6.00%	573.51	2712.21	13183.74	12610.23
	Variability	8514.03	9096.02	9777.87	4.00%	842.19	3697.96	16748.48	15906.29
	Variability	8403.50	8955.08	9587.69	2.00%	1271.30	5100.29	21247.84	19976.54
C499	Nominal	3575.46	3847.00	4192.59	8.98%	405.49	1989.22	9848.17	9442.68
	Variability	3534.69	3788,99	4080.46	6.00%	616.16	2840.73	13190.79	12574.63
	Variability	3494.58	3732.48	4001.41	4.00%	900.32	3896.79	16986.99	16086.67
	Variability	3453.94	3675.59	3923.70	2.00%	1330.04	5392.75	22023.53	20693.49

had at least 8% delay variability. This is a significant amount of variability that might not be tolerable for the given design. We also note that as we reduce this constraint on the delay variability, the nominal leakage as well as the leakage variability increases. Hence, there is a trade-off between delay variability and leakage variability which can be controlled through our proposed formulation.

V. CONCLUSION

In this paper, we presented a general mathematical framework for simultaneous multiplethreshold selection and assignment. Additionally, we also address the problem of fabrication variability and present a variability driven threshold voltage selection and assignment scheme that lets the designer control the leakage and delay variability in the design. An interesting course of future work could be to develop efficient and faster techniques for gate clustering.

REFERENCES

[1] B. J. Sheu, D. L. Scharfetter, P. K. Ko, and M. C. Jeng, "BSIM: Berkeley short-channel IGFET model for MOS transistors," *IEEE J. Solid-State Circuits*, vol. SC–22, no. 4, pp. 558–566, Aug. 1987.

[2] E. M. Sentovich, K. J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. R. Stephan, R. K. Brayton, and A. L. Sangiovanni-Vincentelli, "SIS: A System for sequential circuit synthesis," Dept. Electrical Eng. Comp. Sci., Univ. California, Berkeley, CA, Memo. UCB/ERL M92/41, May 1992.

[3] J. Kao, A. Chandrakasan, and D. Antoniadis, "Transistor sizing issues and tools for multithresh-hold CMOS technology," in *Proc. Design Automation Conf.*, Jun. 1997, pp. 409–414.

[4] M. Anis, S. Areibi, M. Mahmoud, and M. I. Elmasry, "Dynamic and leakage power reduction in MTCMOS circuits using an automated efficient gate clustering technique," in *Proc. Design Automation Conf.*, Jun. 2002, pp. 480–485.

[5] M. Sarrafzadeh, D. A. Knol, and G. E. Tellez, "A delay budgeting algorithm ensuring maximum flexibility in placement," *IEEE Trans. Computer-Aided Design Integr. Circuits Syst.*, vol. 16, no. 11, pp. 1332–1341, Nov. 1997.

[6] Q. Wang and S. B. K. Vrudhula, "Static power optimization of deep submicron CMOS circuits for DUAL V_t technology," in *Proc. Int. Conf. Computer Aided Design*, Nov. 1998, pp. 490–496.

[7] R. X Gu and M. I. Elmasry, "Power dissipation analysis and optimization of deep submicron CMOS digital circuits," *IEEE J. Solid-State Circuits*, vol. 31, no. 5, pp. 703–713, May 1996.

[8] V. Sundararajan and K. Parhi, "Low power synthesis of dual thresh-hold voltage CMOS VLSI circuits," in *Proc. Int. Symp. Low Power Electron. Design*, 1999, pp. 139–144.

Synthesis of Fredkin–Toffoli Reversible Networks

Dmitri Maslov, Gerhard W. Dueck, and D. Michael Miller

***Abstract*—Reversible logic has applications in quantum computing, low power CMOS, nanotechnology, optical computing, and DNA computing. The most common reversible gates are the Toffoli gate and the Fredkin gate. We present a method that synthesizes a network with these gates in two steps. First, our synthesis algorithm finds a cascade of Toffoli and Fredkin gates with no backtracking and minimal look-ahead. Next we apply transformations that reduce the number of gates in the network. Transformations are accomplished via template matching. The basis for a template is a network with m gates that realizes the identity function. If a sequence of gates in the network to be reduced matches a sequence of gates comprising more than half of a template, then a transformation that reduces the gate count can be applied. We have synthesized all three input, three output reversible functions and here compare our results to the optimal results. We also present the results of applying our synthesis tool to obtain networks for a number of benchmark functions.**

***Index Terms*—Design automation, logic design, network optimization, quantum computing.**

I. INTRODUCTION

Reversible logic has attracted significant attention in recent years. It has applications in quantum computing, nanotechnology, low power CMOS, and optical computing. Interest in this area started when Landauer [10] proved that traditional binary irreversible gates lead to power dissipation in a circuit regardless of implementation. Recently, Zhirnov *et al.* [20], among others, estimated that power dissipation in future CMOS (scaled for the year 2016 in accordance with the 2001 International Technology Roadmap for Semiconductors) leads to impossible heat removal, thus suggesting a limit to speeding up CMOS technology

Manuscript received March 18, 2004; revised September 8, 2004. This work was supported in part by grants from the Natural Sciences and Engineering Research Council of Canada.

D. Maslov and D. M. Miller are with the Department of Computer Science, University of Victoria, Victoria, BC V8W 3P6, Canada (e-mail: dmaslov@uvic.ca; mmiller@cs.uvic.ca).

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

Digital Object Identifier 10.1109/TVLSI.2005.844284

devices. Bennett [3] showed that for power not to be dissipated it is necessary that a binary circuit be build from reversible gates. This suggests that reversible technologies and the synthesis of reversible networks are potentially very promising areas of study with regard to further technological advances.

A reversible function (gate) is a bijection. Traditional gates such as AND, OR, and EXOR are not reversible. In fact NOT is the only reversible gate from the traditional set of gates. Several reversible gates have been proposed. Among them are the controlled NOT (also known as the Feynman [7] or the two-bit Toffoli gate), the Toffoli gate [19], and the Fredkin gate [8]. These gates are, perhaps, the most investigated reversible gates. Inexpensive quantum implementations of the Toffoli [2], [11], [17], and the Fredkin [6], [17] gates have been found. While the technological costs of Toffoli and Fredkin gates are comparable, it has been shown [5] that using both types of gates results in network specifications with fewer gates. Therefore, a synthesis procedure exploiting both types of gates is of interest. In this paper, we concentrate on the synthesis of networks with these well-known gates and their straightforward generalizations.

The synthesis of reversible logic differs significantly from traditional irreversible logic synthesis approaches. Fan-outs and loops are not permitted due to the target technology. Outputs from one gate are used as inputs to the next gate resulting in a network that is a cascade of gates.

Only a few synthesis methods have been proposed for reversible logic. Suggested methods include: using Toffoli gates to implement an ESOP (EXOR sum-of-products) [16], using tree search and Reed–Muller expansions [1], exhaustive enumeration [18], heuristic methods that iteratively make the function simpler (simplicity is measured by the Hamming distance [4] or by spectral means [14]), and transformation based synthesis [9], among others. Some methods use excessive search time, others are not guaranteed to converge, and some require many additional *garbage* outputs. In this paper, we employ the following definition of garbage: outputs that are not required by the function specification, but appear in a design due to the reversibility requirements [12]).

We follow the two-step approach used in [15] for Toffoli network synthesis. Our method first finds an initial network with no backtracking and minimal look-ahead. We first present a unidirectional algorithm and then show how reversibility leads to a more effective bidirectional algorithm. Both the unidirectional and the bidirectional algorithms always find a solution. The second step of our approach is to apply a set of template transforms that reduce the size of the network. In this paper we describe and classify the templates used for such transformations.

II. Definitions

We use generalized Toffoli [19] and generalized Fredkin [8] gates defined as follows.

Definition 1: For the set of domain variables $\{x_1, x_2, \ldots, x_n\}$ the **generalized Toffoli gate** has the form $\mathrm{TOF}(C;T)$, where $C = \{x_{i_1}, x_{i_2}, \ldots, x_{i_k}\}, T = \{x_j\}$ and $C \cap T = \emptyset$. It maps each pattern $(x_1^+, x_2^+, \ldots, x_n^+)$ to $(x_1^+, x_2^+, \ldots, x_{j-1}^+, x_j^+ \oplus x_{i_1}^+ x_{i_2}^+ \ldots x_{i_k}^+, x_{j+1}^+, \ldots, x_n^+)$.

Definition 2: For the set of domain variables $\{x_1, x_2, \ldots, x_n\}$ the **generalized Fredkin gate** has the form $\mathrm{FRE}(C;T)$, where $C = \{x_{i_1}, x_{i_2}, \ldots, x_{i_k}\}, T = \{x_j, x_l\}$ and $C \cap T = \emptyset$. It maps each pattern $(x_1^+, x_2^+, \ldots, x_n^+)$ to $(x_1^+, x_2^+, \ldots, x_{j-1}^+, x_l^+, x_{j+1}^+, \ldots, x_{l-1}^+, x_j^+, x_{l+1}^+, \ldots, x_n^+)$ iff $x_{i_1}^+ x_{i_2}^+ \ldots x_{i_k}^+ = 1$, otherwise the pattern is unchanged. In other words, the generalized Fredkin gate interchanges x_j and x_l if, and only if, the product of the variables in C equals 1.

For both gate types, C will be called the **control** set and T will be called the **target** set. The number of elements in the set of controls C defines the **width** of the gate. The set of generalized Toffoli and generalized Fredkin gates will be called the **Fredkin–Toffoli family**. For the control set $C = \{x_3, x_4, \ldots, x_{k+2}\}$ the pictorial representation of gate $\mathrm{TOF}(C; x_2)$ is shown in Fig. 1(a), and the pictorial representation of gate $\mathrm{FRE}(C; x_1, x_2)$ is shown in Fig. 1(b).

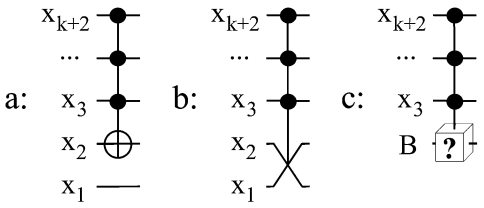

Fig. 1. Toffoli and Fredkin gates, and the box notation.

Toffoli and Fredkin gates are closely related. In fact, they can be written as one general gate $G(S;B)$. Section IV illustrates how useful it is to unite these two gate types. The uniform way of writing Toffoli and Fredkin gates is captured in the definition of a **box notation** $G(S;B)$, which for $|B| = 1$ is the $\mathrm{TOF}(S;B)$ gate and for $|B| = 2$ is $\mathrm{FRE}(S;B)$. This way of writing the gates is needed when we consider a gate from the Fredkin–Toffoli family and do not want to specify its type. If the size of the set B is not specified, it can be either 1 or 2. The gate shown in Fig. 1(c) is $G(C;B)$ where the set B is not specified.

III. Unidirectional and Bidirectional Algorithms

Unidirectional Algorithm: The unidirectional algorithm assumes that the function to be synthesized is given as a truth table, and starts the synthesis process with an empty network. The algorithm transforms each row of the output specification to the corresponding input pattern assigning gates from the Fredkin–Toffoli family building the network in order from the outputs to the inputs.

Step 0: Use NOT gates as required to transform the first output pattern to the form $(0, 0, \ldots, 0)$.

Let $(b_1, b_2, \ldots, b_n)$ be the first output pattern. To bring it to the form $(0, 0, \ldots, 0)$ requires gates $\mathrm{TOF}(\emptyset; x_i)$ for every $i : b_i \neq 0$. After adding the gates to the cascade, update the output part of the table by applying the appropriate inversions to all output patterns.

Step S, $1 \leq S \leq 2^n - 2$*:* Without influencing the patterns of lower order that were transformed in previous steps of the algorithm, use the least number of gates with the fewest possible controls (those are usually less costly when implemented in any particular technology) to bring the output pattern to the form of the corresponding input pattern.

Assume pattern $(a_1, a_2, \ldots, a_n)$ is the binary representation of S. Let $(b_1, b_2, \ldots, b_n)$ be the output pattern corresponding to $(a_1, a_2, \ldots, a_n)$. $(b_1, b_2, \ldots, b_n)$ must be equal to $(a_1, a_2, \ldots, a_n)$ or must be the binary representation of a value greater than S since all values less than S correspond to earlier entries in the truth table which are already in place. If the patterns are equal, no action is required.

When transforming $(b_1, b_2, \ldots, b_n)$ to $(a_1, a_2, \ldots, a_n)$, note that each application of a Toffoli gate is capable of flipping one bit of pattern $(b_1, b_2, \ldots, b_n)$, and each Fredkin gate is capable of interchanging a pair of unequal Boolean values.

Now, the problem can be formulated as follows: using the two operations "flip" and "swap" bring the Boolean pattern $(b_1, b_2, \ldots, b_n) \succ (a_1, a_2, \ldots, a_n)$ to the form $(a_1, a_2, \ldots, a_n)$ so that all intermediate Boolean patterns are greater than $(a_1, a_2, \ldots, a_n)$. The controls for the corresponding gates will be assigned later. The solution is as follows.

- If the number of ones in $(b_1, b_2, \ldots, b_n)$ is less than the number of ones in $(a_1, a_2, \ldots, a_n)$ apply "swaps"

that improve 2 bit positions and then flip the remaining incorrect bits. Use "swaps" so that the order of each intermediate pattern $(x_1, x_2, \ldots, x_n)$ is greater than the order of $(a_1, a_2, \ldots, a_n)$. This can be easily done if "swaps" are used on the low order bits first. Note, that the initial pattern $(b_1, b_2, \ldots, b_n)$ is of an order higher than $(a_1, a_2, \ldots, a_n)$, while having lesser number of ones in it. Thus, the position of the most significant 1 bit of $(b_1, b_2, \ldots, b_n)$ is greater than (further left) the position of the most significant 1 bit of $(a_1, a_2, \ldots, a_n)$. Therefore, it is used as the control (when a control is needed) for all corresponding Fredkin and Toffoli gates except the last Toffoli gate, for which the control consists of all variables which are 1 in $(a_1, a_2, \ldots, a_n)$.

- If the number of ones in $(b_1, b_2, \ldots, b_n)$ is equal to the number of ones in $(a_1, a_2, \ldots, a_n)$, it is possible to transform one pattern into the other using "swap" operation only. The set of controls while considering an intermediate pattern $(x_1, x_2, \ldots, x_n)$ is taken as a minimal subset of its one values such that this subset forms a Boolean pattern of an order higher than $(a_1, a_2, \ldots, a_n)$.
- If the number of ones in $(b_1, b_2, \ldots, b_n)$ is greater than the number of ones in $(a_1, a_2, \ldots, a_n)$, apply "swaps" starting from the right end of the pattern $(b_1, b_2, \ldots, b_n)$ and then apply the necessary Toffoli gates. The controls can be found using the procedures described in the above cases.

Step $2^n - 1$*:* When the first $2^n - 1$ patterns are properly aligned, the last pattern will by definition be correct.

Bidirectional Algorithm: The unidirectional synthesis algorithm works from the output to input by adding gates in one direction only. We now describe an alternative that performs the matching by reordering rows of the truth table. It adds gates from the inputs toward the outputs.

Toffoli Gate Application: Without loss of generality consider gate $\mathrm{TOF}(C; x_{k+1}), C = \{x_1, x_2, \ldots, x_k\}$, with the controls on the first k variables and target on variable $k+1$. Then, in the input part of the truth table the patterns $(1, 1, \ldots, 1, x^0_{k+1}, x^0_{k+2}, \ldots, x^0_n)$ and $(1, 1, \ldots, 1, \bar{x}^0_{k+1}, x^0_{k+2}, \ldots, x^0_n)$ will be interchanged. This is the same as permuting the output patterns associated with input patterns $(1, 1, \ldots, 1, x^0_{k+1}, x^0_{k+2}, \ldots, x^0_n)$ and $(1, 1, \ldots, 1, \bar{x}^0_{k+1}, x^0_{k+2}, \ldots, x^0_n)$ without reordering the input side of the table.

Fredkin Gate Application: Applying the Fredkin gate $\mathrm{FRE}(C; x_{k+1}, x_{k+2}), C = \{x_1, x_2, \ldots, x_k\}$ results in the following change of all patterns in the input part of the table: $(1, 1, \ldots, 1, x^0_{k+1}, x^0_{k+2}, x^0_{k+3}, \ldots, x^0_n)$ is interchanged with $(1, 1, \ldots, 1, x^0_{k+2}, x^0_{k+1}, x^0_{k+3}, \ldots, x^0_n)$. This operation is equivalent to interchanging the output patterns of the truth table associated with the input patterns $(1, 1, \ldots, 1, x^0_{k+1}, x^0_{k+2}, x^0_{k+3}, \ldots, x^0_n)$ and $(1, 1, \ldots, 1, x^0_{k+2}, x^0_{k+1}, x^0_{k+3}, \ldots, x^0_n)$. Since gates identified in this procedure apply to input patterns, they are added in the order identified from the input side of the cascade.

Given the above and a procedure similar to that described for the unidirectional algorithm, it is possible to move an output pattern to its correct position, i.e., so that it matches the input pattern by row interchange.

Given the unidirectional algorithm and the alternative just described, our bidirectional synthesis algorithm proceeds as follows. For each row of the truth table in order from $(0, 0, \ldots, 0)$, it either transforms the output pattern to the input as described in the unidirectional algorithm, or it performs the match by reordering rows of the truth table whichever requires less gates. If both require the same number of gates, the gates identified in the unidirectional approach are used.

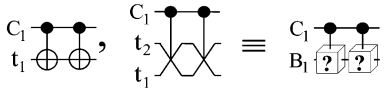

Fig. 2. Duplicate deletion rule.

IV. Template Simplification Tool

Let a **size** m **template** be a sequence of m gates (a network) that realizes the identity function. The template must be independent of smaller size templates, 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. For a template $G_0 G_1 \ldots G_{m-1}$ its **application** is one of the two operations:

Forward Application: A sequence of gates in the network which matches the sequence $G_i G_{(i+1) \bmod m} \ldots G_{(i+k-1) \bmod m}$ in the template $G_0 G_1 \ldots G_{m-1}$ is replaced with the sequence $G_{(i-1) \bmod m} G_{(i-2) \bmod m} \ldots G_{(i+k) \bmod m}$, where $k \in M, k \geq \frac{m}{2}$.

Reverse Application: A sequence of gates in the network which matches the sequence $G_i G_{(i-1) \bmod m} \ldots G_{(i-k+1) \bmod m}$ is replaced with the sequence $G_{(i+1) \bmod m} G_{(i+2) \bmod m} \ldots G_{(i-k) \bmod m}$, where $k \in M, k \geq m/2$.

It can be shown that neither forward or reverse template application as described changes the output of the network. Each application of a template of size m for parameter $k > m/2$ leads to a reduction in the number of gates.

A classification scheme is used to reduce the number of different templates. The classes reported here were found by hand and were verified to be complete by exhaustive enumeration.

Definition 3: A **template class** is defined by a set of disjoint formulas, i.e., formulas $G_1(S_1, B_1), G_2(S_2, B_2), \ldots, G_m(S_m, B_m)$, where the following occurs.

- Depending on the number of elements in B_i, G_i is written as TOF (for $|B_i| = 1$) or FRE (for $|B_i| = 2$).
- S_i is written as a union of sets (C) and single variables (t) : $S_i = C_{i_1} + C_{i_2} + \cdots + C_{i_k} + t_{i_1} + t_{i_2} + \cdots + t_{i_j}$.
- If $|B_i| = 1$, it is written as a single variable, t_j; if $|B_i| = 2$ it is written as the union $t_j + t_k$.
- The C_i sets are mutually disjoint, the t variables are distinct and do not appear in any C_i.

In order to classify the templates, we need to discuss the box notation in more detail. If a box appears in a template, there are certain rules for changing the template network when the operation of EXOR or SWAP is assigned to the box. If the assignment is EXOR, then the box is substituted with the EXOR symbol as are all other box symbols on the same line. If the assignment was SWAP, the line with the box becomes the two lines, on which the SWAP acts. Every occurrence of a control on the line with this box is substituted with two controls and every occurrence of the box symbol is substituted with SWAP. Lastly, if a box symbol in a template is not specified, it can be either EXOR or SWAP which are substituted into the template by the above rules.

$\mathbf{m = 1}$. Clearly, there are no templates of size 1, since no single gate realizes the identity.

$\mathbf{m = 2}$. There is one class of templates of size 2, the **duplicate deletion rule** (Fig. 2). This class is a generalization of the Toffoli gate duplicate deletion rule [15]. It is true for any gate type which perform a self-inverse transformation. In gate-specific notation, this class can be written as two formulas, one for two Toffoli gates and one for two Fredkin gates: $\mathrm{TOF}(C_1, t_1)\mathrm{TOF}(C_1, t_1)$ and $\mathrm{FRE}(C_1, B_1)\mathrm{FRE}(C_1, B_1)\,(B_1 = t_1 + t_2)$ as shown in Fig. 2. As shown, it can be written as one template using the box notation.

$\mathbf{m = 3}$. There are no templates of size 3.

$\mathbf{m = 4}$. There are three classes of size 4 templates. A very important class is the **passing rules** (analogy of the passing rule from [15]). It

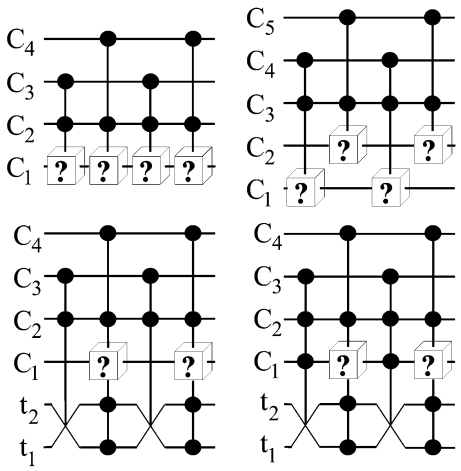

Fig. 3. Passing rule class.

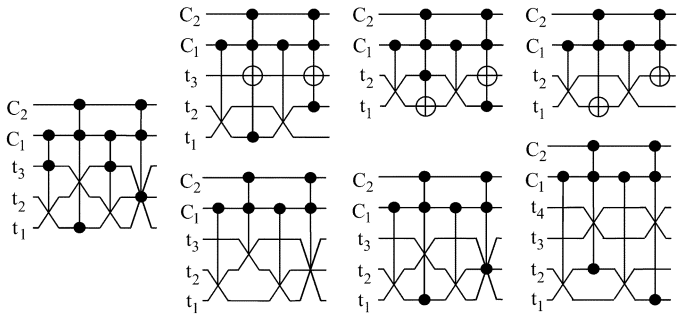

Fig. 4. Group of semi passes.

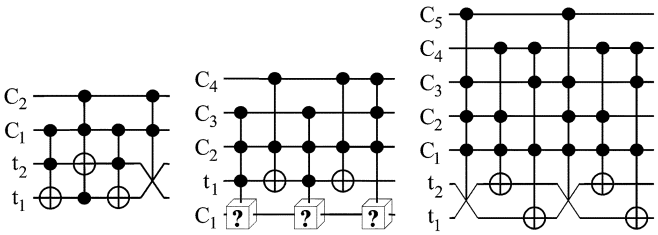

Fig. 5. Fredkin definition class, class of size 5, and class of size 6.

defines when the order of the two gates in a network can be changed. All templates defining passing rules are shown in Fig. 3.

The second class is the *semi-passing rules* (Fig. 4). Its main features are the equality of the first and third gates, and the fourth gate being the second gate transformed by the target of the first gate.

The third class, which we call *Fredkin definition* (Fig. 5), is a generalization of the popular network simulating Fredkin gate with Toffoli gates [17].

$\mathbf{m = 5}$. There is only one class of templates of size 5 (Fig. 5). Even though longer matches are required, it turns out that in practice this class is the most useful.

$\mathbf{m = 6}$. There is one class of templates of size 6 (Fig. 5).

V. Results

We have written a C++ program that synthesizes networks using the bidirectional algorithm and then applies the template tool. We ran our program for all reversible functions with 3 three variables. The column labeled **BISYNTH** in Table I shows how many of these functions can be realized with $k = 0 \ldots 9$ NOT, CNOT, Toffoli, SWAP and Fredkin gates. It took 79.69 CPU seconds on an AMD Athlon 2400+ XP computer with 512 M of RAM to synthesize these 40 320 functions.

TABLE I
Number of Reversible Functions Using a Specified Number of Gates for $n = 3$

Size	BISYNTH	NCTSF	NCT
0	1	1	1
1	18	18	12
2	184	184	102
3	1290	1318	625
4	5680	6474	2780
5	13209	17695	8921
6	13914	14134	17049
7	5503	496	6817
8	512	0	32
9	9	0	0
WA:	5.437	5.134	5.866

TABLE II
Results of Benchmark Function Synthesis

name	in	out	size	Toffoli	Fredkin-Toffoli
3_17	3	3	3	6	6^t
4_49	4	4	4	16	13^{ft}
4mod5	4	1	5	8	8^t
5mod5	5	1	6	17	10^t
add3	3	2	4	4	4^t
ham3	3	3	3	5	4^{ft}
ham7	7	7	7	23	23^t
ham15	15	15	15	132	132^t
hwb4	4	4	4	17	11^f
hwb5	5	5	5	55	24^f
hwb6	6	6	6	126	65^f
hwb7	7	7	7	289	166^f
rd53	5	3	7	12	12^t
cycle2_10	12	12	12	19	19^t
cycle3_17	20	20	20	48	48^t

By exhaustive search, we have enumerated the optimal realizations for these functions. The results are given in the column labeled **NCTSF**. The average for the optimal case is 5.134 whereas the average for our method is 5.437, 5.9% higher.

To show the advantage of allowing Fredkin gates, we include the optimal results for NOT, CNOT and Toffoli gates given in [18] as column **NCT**. The average in this case is 5.866 which is 14.3% higher than the optimal when Fredkin and SWAP gates are permitted.

A. Benchmarks

The algorithm does not in theory impose any limits on the number of variables. However, since the truth table must be stored, functions with more than 20 variables may require too much space and time. In particular, using the computer mentioned above it took approximately 25 minutes to synthesize the network for the largest specification, $\mathrm{cycle3\_17}$ which has 20 lines. Runtimes for smaller specifications are noticeably shorter, e.g., 8.9 seconds for the second largest function (15 lines.) For initially irreversible functions, we created a reversible specification using methods discussed in [12], [14].

Our benchmark results are summarized in Table II. The **name**, **in**, **out**, and **size** columns contain the name of a benchmark function, the original number of inputs, the original number of outputs and the size, the number of inputs and outputs, of a minimal reversible specification [12]. Function specifications as well as the networks that we reported in this paper can be found on the web [13]. The **Toffoli** and **Fredkin–Toffoli** columns show first the number of gates used in our designs when only Toffoli gates are used, and second the number of gates when Fredkin gates are also allowed. Superscripts in the last column

indicate which gates are present in the design: "t" stands for Toffoli gates only, "f" for Fredkin gates only, and "ft" indicates when the network contains both Fredkin and Toffoli gates.

It is interesting to note that Fredkin gates may not help the synthesis $(\mathrm{ham15})$, but sometimes help to reduce the size of the network significantly $(\mathrm{hwb5})$. Also, sometimes when Fredkin gates are allowed, the synthesized network may not contain Fredkin gates and be smaller than a network synthesized with the condition that Fredkin gates are not allowed $(5\mathrm{mod}5)$. This occurs because Fredkin gates are included in the initial synthesis but then removed during the template application.

Three of the functions used here were used as benchmarks in [1]. It is possible to make a fair comparison of the results, since our networks for these functions use Toffoli gates only. Both our method and the one reported in [1] synthesized function $\mathrm{add3}$ with four gates. Further, this 4-gate realization is believed to be optimal. Both methods synthesized function $\mathrm{rd53}$ with 12 gates. However, the network reported in [1] is incorrect. Finally, we synthesized $\mathrm{cycle2\_10}$ with 19 gates, while [1] presented a network with 26 gates.

VI. Conclusion

In this paper we presented a two-step method for the synthesis of reversible networks using Toffoli and Fredkin gates. Our approach uses minimal garbage.

Applying our approach to all three variable reversible functions yielded results which are on average 5.9% above the optimal. We also presented networks for a number of benchmark problems which show the advantage of allowing Fredkin gates in addition to Toffoli gates.

For three variable functions, the results reported here show roughly the same type of performance as earlier work using NOT, CNOT, and Toffoli gates alone. For example, Agrawal and Jha [1] report an average of 6.10 whereas the optimal is 5.866. However, we have shown benchmarks where our method found a purely Toffoli network superior to the one reported in [1].

Future work will improve the efficiency of our approach by replacing truth tables with a more compact representation. We are also investigating better heuristics in the synthesis procedure and alternative methods for applying the templates.

References

[1] A. Agrawal and N. K. Jha, "Synthesis of reversible logic," *Proc. DATE*, pp. 21 384–21 385, Feb. 2004.

[2] 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," *Phys. Rev. A*, vol. 52, pp. 3457–3467, 1995.

[3] C. H. Bennett, "Logical reversibility of computation," *IBM J. Res. Development*, vol. 17, pp. 525–532, Nov. 1973.

[4] G. W. Dueck and D. Maslov, "Reversible function synthesis with minimum garbage outputs," in *Proc. 6th Int. Symp. Representations and Methodology of Future Computing Technologies*, Mar. 2003, pp. 154–161.

[5] G. W. Dueck, D. Maslov, and D. M. Miller, "Transformation-based synthesis of networks of Toffoli/Fredkin gates," in *Proc. IEEE Canadian Conf. Electrical and Computer Engineering*, May 2003, pp. 211–214.

[6] X. Fei, D. Jiang-Feng, S. Ming-Jun, Z. Xian-Yi, H. Rong-Dian, and W. Ji-Hui, "Realization of the Fredkin gate by three transition pulses in a nuclear magnetic resonance quantum information processor," *Chinese Phys. Lett.*, vol. 19, no. 8, pp. 1048–1050, 2002.

[7] R. Feynman, "Quantum mechanical computers," *Opt. News*, vol. 11, pp. 11–20, 1985.

[8] E. Fredkin and T. Toffoli, "Conservative logic," *Int. J. Theoretical Phys.*, vol. 21, pp. 219–253, 1982.

[9] K. Iwama, Y. Kambayashi, and S. Yamashita, "Transformation rules for designing CNOT-based quantum circuits," *Proc. DAC*, pp. 419–424, Jun. 2002.

[10] R. Landauer, "Irreversibility and heat generation in the computing process," *IBM J. Res. Develop.*, vol. 5, pp. 183–191, 1961.

[11] D. Maslov and G. Dueck, "Improved quantum cost for n-bit Toffoli gates," *IEE Electron. Lett.*, vol. 39, no. 25, pp. 1790–1791, Dec. 2003.

[12] D. Maslov and G. W. Dueck, "Garbage in reversible design of multiple output functions," in *Proc. 6th Int. Symp. Representations and Methodology of Future Computing Technologies*, Mar. 2003, pp. 162–170.

[13] D. Maslov, N. Scott, and G. W. Dueck. (2004, Aug.) Reversible logic synthesis benchmarks page. [Online]. Available: http://www.cs.uvic.ca/~dmaslov/

[14] D. M. Miller and G. W. Dueck, "Spectral techniques for reversible logic synthesis," in *Proc. 6th Int. Symp. Representations and Methodology of Future Computing Technologies*, Mar. 2003, pp. 56–62.

[15] D. M. Miller, D. Maslov, and G. W. Dueck, "A transformation based algorithm for reversible logic synthesis," *Proc. DAC*, pp. 318–323, Jun. 2003.

[16] A. Mishchenko and M. Perkowski, "Logic synthesis of reversible wave cascades," *Proc. IWLS*, pp. 197–202, Jun. 2002.

[17] M. Nielsen and I. Chuang, *Quantum Computation and Quantum Information*. Cambridge, U.K.: Cambridge Univ. Press, 2000.

[18] V. V. Shende, A. K. Prasad, I. L. Markov, and J. P. Hayes, "Synthesis of reversible logic circuits," *IEEE Trans. Computer-Aided Design Integr. Syst.*, vol. 22, no. 6, pp. 723–729, Jun. 2003.

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

[20] V. V. Zhirnov, R. K. Cavin, J. A. Hutchby, and G. I. Bourianoff, "Limits to binary logic switch scaling—A gedanken model," *Proc. IEEE*, vol. 91, no. 11, pp. 1934–1939, Nov. 2003.