Transformation-based Synthesis of Networks of Toffoli/Fredkin Gates

Gerhard W. Dueck, Dmitri Maslov
Faculty of Computer Science
University of New Brunswick
Fredericton, N.B. E3B 5A3
gdueck@unb.ca, dmaslov@unb.ca

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

Abstract

Reversible logic has attracted significant attention in recent years. It has applications in low power CMOS, quantum computing, nanotechnology, and optical computing. 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 gate), the Toffoli gate, and the Fredkin gate. An n-input Toffoli gate 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'. An n-input Fredkin gate has n-2 control lines which pass through the gate unaltered and two target lines on which the values are swapped if all the control lines have value '1'. A NOT gate is the special case of a Toffoli gate with no control inputs. Likewise, a SWAP gate is the special case of a Fredkin gate with no control inputs. In this paper, we will review a transformation-based synthesis procedure targeted to Toffoli gates and show how it can be extended to allow Fredkin gates. This extension results in circuits with fewer gates.

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. This results in a high degree of interdependence among gates. Our algorithm first finds an initial circuit with no backtracking and minimal look-ahead. We exploit reversibility directly in our synthesis approach. This method always finds a solution. Next we apply a set of template transforms that reduce the size of the circuit. We synthesize all three input, three output reversible functions and compare our results to those obtained previously.

Keywords: *Reversible Logic, Synthesis, Toffoli Gates, Fredkin Gates, Templates.*

1 Definitions

We consider cascades of generalized Toffoli and generalized Fredkin gates as defined below.

Definition 1. For the set of domain variables $\{x_1, x_2, ..., x_n\}$ a **generalized Toffoli gate** has the form $TOF(C;T)$, where $C = \{x_{i_1}, x_{i_2}, ..., x_{i_k}\}$, $T = \{x_j\}$ and $C \cap T = \emptyset$. Such a gate maps the Boolean pattern $(x_1^0, x_2^0, ..., x_n^0)$ to $(x_1^0, x_2^0, ..., x_{j-1}^0, x_j^0 \oplus x_{i_1}^0 x_{i_2}^0 ... x_{i_k}^0, x_{j+1}^0, ..., x_n^0)$. The set C which controls the change of the j-th bit is called the set of **control lines** and T is called the **target**. ▶

Several gates of this family are well-known and widely studied. $TOF(\emptyset; x_1)$, or simply $TOF(x_1)$ is the special case where there are no control inputs, so x_1 is always inverted, *i.e.* it is a NOT gate. $TOF(x_1; x_2)$ has been termed the Feynman [2] or controlled-NOT gate (CNOT). $TOF(x_1, x_2; x_3)$ is often referred to simply as a Toffoli gate [7].

Definition 2. For the set of domain variables $\{x_1, x_2, ..., x_n\}$ the **generalized Fredkin gate** has the form $FRE(C;T)$, where $C = \{x_{i_1}, x_{i_2}, ..., x_{i_k}\}, T = \{x_j, x_l\}$ and $C \cap T = \emptyset$ and maps the Boolean pattern $(x_1^0, x_2^0, ..., x_n^0)$ to $\{x_1^0, x_2^0, ..., x_{j-1}^0, x_l^0, x_{j+1}^0, ..., x_{l-1}^0, x_j^0, x_{l+1}^0, ..., x_n^0)$ iff $x_{i_1} x_{i_2} ... x_{i_k} = 1$ otherwise all bits are left unchanged. In other words, the generalized Fredkin gate interchanges bits x_j and x_l iff corresponding product equals 1. ▶

The SWAP gate $FRE(\emptyset; x_1, x_2)$ is the case of a Fredkin family gate with no controls. Although in some technologies SWAPs are done at no cost, we assume that there is a cost associated with such a gate. Another example of a gate from the Fredkin family, is the original Fredkin gate $FRE(x_1; x_2, x_3)$ [3]. It has a single control.

2 Optimal Synthesis

Given a reversible function, we want to find a network realization composed of Toffoli and Fredkin gates. Synthesis methods for Toffoli gate networks have been proposed in [1], [4], [5], [6]. Markov *et al.* [6] found all optimal networks for three input reversible functions by matching all the minimal Toffoli gate networks (networks made of gates NOT, CNOT, Toffoli) with all reversible functions of three variables. They also considered minimal networks for three input reversible functions with NOT, CNOT, Toffoli and SWAP gates. The SWAP is part of a different family, so it is interesting to expand the results of optimal synthesis to include Fredkin gates.

Size	NCT	NCTS	NCTSF
0	1	1	1
1	12	15	18
2	102	134	184
3	625	844	1318
4	2780	3752	6474
5	8921	11194	17695
6	17049	17531	14134
7	10253	6817	496
8	577	32	0
WA	5.867	5.629	5.134

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

Table 1 shows the number of three input reversible functions requiring the specified number of gates for different gate type sets. The values in NCT (networks with NOT, CNOT, and Toffoli gates) and NCTS (NTC plus SWAP gates) are taken from Markov *et al.*. Column CNTSF adds Fredkin gates and gives the results produced by a program we wrote. The weighted average (shown in the bottom row labeled WA) gives the average cost of a three input function. Clearly, it is beneficial to include Fredkin gates, since the weighted average drops significantly, and no function requires more than 7 gates.

3 Regular Synthesis

In earlier work [5], we proposed a synthesis method that produces networks of generalized Toffoli gates for any given reversible specification. Generalized Fredkin gates can easily be incorporated into the algorithm.

For the algorithm in [5], it was easy to construct the worst case scenario function. In particular, for $n = 3$ such a function was constructed (it is unique) and called 3_17*.pla*. The cost of realizing this function with the naive algorithm from [5] is 17 gates.

Example 1. Applying the basic method for the 3_17*.pla* function shown in Table 2.

- Step 0. The output pattern corresponding to the input pattern $(0, 0, 0)$ is $(1, 1, 1)$. In order to bring it to the form $(0, 0, 0)$ apply 3 NOT's: $TOF(; a)$, $TOF(; b)$ and $TOF(; c)$. Update the table (Table 2, **S1**) to show the new output pattern.
- Step 1. For input pattern $(0, 0, 1)$ we have the output pattern $(1, 1, 0)$. In order to match the last two bits to the input, swap bits b and c by $FRE(; b, c)$ and then use $TOF(c; a)$ to bring the "swapped" pattern $(1, 0, 1)$ to the form $(0, 0, 1)$. Neither of the gates used changes anything of the order less than $(0, 0, 1)$. Also note, that this is not a unique way of changing the output pattern to match the input pattern even for the smallest set of controls. $FRE(; a, c)$, $TOF(c; b)$ would do the same job.
- Step 2. The next input pattern, $(0, 1, 0)$ does not match the correspondent output pattern, $(1, 1, 1)$ (Table 2, **S2**). Apply the gates $TOF(b; c)$, $TOF(b; a)$ to make the match.
- Step 3. Apply $TOF(a; c)$ and $FRE(c; a, b)$ to match the output pattern $(1, 0, 0)$ of Table 2, **S2** to the desired input pattern $(0, 1, 1)$.
- Step 4. Use $TOF(a; c)$ and $TOF(a; b)$ to bring $(1, 1, 1)$ (Table 2, **S4**) to the form $(1, 0, 0)$.
- Step 5. Finally, use $FRE(a; b, c)$ to transform $(1, 1, 0)$ from Table 2, **S5** to $(1, 0, 1)$.
- Steps 6,7 are empty since the output completely matches the input (Table 2, **S6**).

In	**Out**	**S1**	**S2**	**S3**	**S4**	**S5**	**S6**
000	111	**000**	**000**	**000**	**000**	**000**	**000**
001	001	110	**001**	**001**	**001**	**001**	**001**
010	100	011	111	**010**	**010**	**010**	**010**
011	011	100	100	100	**011**	**011**	**011**
100	000	111	011	110	111	**100**	**100**
101	010	101	110	011	101	110	**101**
110	110	001	010	111	110	101	**110**
111	101	010	101	101	100	111	**111**

Table 2: Basic approach synthesis.

The resulting circuit has 12 gates as opposed to 17 for the naive approach with Toffoli gates only. The circuit is illustrated in Fig. 1(a).

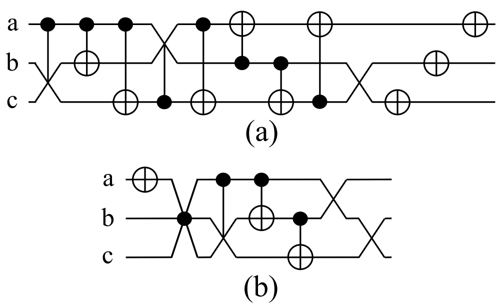

Figure 1: Circuits.

In	**Out**	**S1**	**S2**	**S3**	**S4**	**S5**	**S6**
000	111	**000**	**000**	**000**	**000**	**000**	**000**
001	001	010	**001**	**001**	**001**	**001**	**001**
010	100	110	101	**010**	**010**	**010**	**010**
011	011	101	110	111	**011**	**011**	**011**
100	000	111	111	110	110	**100**	**100**
101	010	001	010	100	100	110	**101**
110	110	100	100	011	111	101	**110**
111	101	011	011	101	101	111	**111**

Table 3: Bidirectional synthesis.

Example 2. Use the bidirectional algorithm to build a circuit for 3_17.*pla*.

- Step 0. We can match the output pattern $(1,1,1)$ with the input pattern $(0,0,0)$ by assigning $TOF(;a)$ to the beginning of the cascade. This transformation interchanges the output patterns in front of input patterns $(0,\alpha,\beta)$ and $(1,\alpha,\beta)$ resulting in the output shown in Table 3, **S1**.

- Step 1. To change $(0,1,0)$ to the form $(0,0,1)$ swap the last two bits (use $FRE(;b,c)$) at the end of the cascade.

- Step 2. To change $(1,0,1)$ in Table 3, **S2** to the form $(0,1,0)$ one gate is not enough. A few choices are possible at this step. Apply gates $FRE(;a,b)$ and $TOF(b;c)$, both at the end of the cascade.

- Step 3. The two gates $FRE(b;a,c)$ assigned at the beginning of the network and $TOF(b,c;a)$ at the end are doing absolutely the same change and both bring target pattern $(1,1,1)$ of Table 3, **S3** to the desired form $(0,1,1)$. We choose $FRE(b;a,c)$.

- Step 4. Pattern $(1,1,0)$ can be brought to the form $(1,0,0)$ by using the gate $TOF(a;b)$ at the end of the cascade.

- Step 5. Gate $FRE(a;b,c)$ assigned to the end of the cascade or to the beginning of cascade makes the same change - it brings $(1,1,0)$ to the desired form $(1,0,1)$. Since this is the last step (column **S6** matches the input column exactly), the gate chosen can be viewed as either arising from the input or the output side since for any circuit the last element of the part built from its beginning is the first element of the cascade part built from its end. In other words, the two parts of the cascade meet at gate $FRE(a;b,c)$.

- Steps 6,7 are empty.

The cascade consists of 7 gates, and the circuit is shown in Figure 1(b).

A template simplification tool was introduced in [5], since the network created by the algorithm is usually not optimal. The idea of a template is to replace a sequence of gates with an equivalent shorter sequence. Further, we extend the template simplification tool to include Fredkin gates.

The template definition is taken from [**?**]. Let a **size** m **template** be a sequence of m gates $(G_0\ G_1 ...\ G_{m-1})$ which realizes the identity function. Any template of size m must be independent of templates of smaller size, *i.e.* for a given template size m no application of any set of templates of smaller size can decrease the number of its elements. **Application** of the template $G_0\ G_1 ...\ G_{m-1}$ is one of the following two operations:

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

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

The introduction of Fredkin gates adds several new templates to those considered in [**?**] and [5]. The new templates are shown in Figure 2. In addition to these

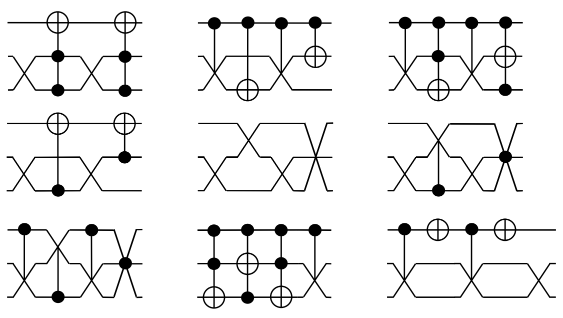

Figure 2: New templates for 3 input functions.

Size	NCT	NCTS	NCTS*	NCTSF	NCTSF*
0	1	1	1	1	1
1	12	15	15	18	18
2	102	134	130	184	175
3	625	844	767	1318	1105
4	2780	3752	2981	6474	4437
5	8921	11194	7518	17695	10595
6	17049	17531	12076	14134	13606
7	10253	6817	11199	496	8419
8	577	32	4726	0	1877
9	0	0	792	0	86
10	0	0	110	0	1
11	0	0	5	0	0
WA:	5.867	5.629	6.176	5.134	5.724

Table 4: Results

templates we use the following two rules. **Duplication deletion rule:** if at any time two adjacent gates are equal, they can be deleted. **Moving rule:** two gates can be interchanged if controls of one do not intersect with the target of the other or, if one of the gates is a Fredkin gate, when both of the Fredkin gate targets are in the controls of the other gate.

In order to check whether we have all the templates for $n = 3$ of size 5 and less, we created a program that finds all the circuits that realize identity function of size 5 and less and try to apply the templates that are listed. The program finds occurrence of one of the templates we show for each case, so we conclude that we have found all templates of size 5 and less.

4 Experimental Results

We developed a program that runs a version of the algorithm (bidirectional with input reduction and permutations) and uses the template tool. At this point, our program uses the 8 four gate templates shown in Figure 2. The size 5 template is not used by our program and we believe that when we incorporate it, there will be a further reduction in the network size. Another reduction that can be easily done is based on the fact that if the two functions, f and f^{-1} have networks of a different size, one can create a network for f by applying the gates for f^{-1} in reverse order.

Even though several simplification operations have not been realized yet, the results from our program are surprisingly close to the optimal (for $n = 3$).

To summarize the results, we created a Table 4. Compare our results (column NCTSF*) to the optimal (column NCTSF) and our previous results for the model gates NOT, CNOT, Toffoli and SWAP (column NCTS*) [5].

Using Fredkin gates in conjunction with our synthesis method is beneficial, since the corresponding weighted average for the regular synthesis with NOT, CNOT, Toffoli, SWAP, and Fredkin is better than the weighted average of optimal synthesis for NOT, CNOT, and Toffoli (column NCT).

Acknowledgement

This work was supported in part by research grants from the Natural Sciences and Engineering Research Council of Canada. This work was completed while the last author was on sabbatical at the University of New Brunswick.

References

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

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

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

[4] 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.

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

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

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