Synthesis of Fredkin-Toffoli Reversible Networks

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

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 synthesize 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 size of the circuit. 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 in the network to be synthesized matches more than half of a template, then a transformation that reduces the gate count can be applied.

We synthesize all three input, three output reversible functions and compare our results to those previously obtained to test the quality of our synthesis approach. We also use our synthesis tool to obtain circuits for some benchmark functions.

Keywords: Logic design, Design automation, Circuit optimization, Quantum theory.

I. INTRODUCTION

Reversible logic has attracted significant attention in recent years. It has applications in quantum computing, nanotechnology, low power CMOS, and optical computing. It started when Landauer [10] proved that traditional binary irreversible gates lead to power dissipation in a circuit regardless of implementation. Recently, Zhirkov et al. [20], among other results, calculated 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, and thus suggesting the impossibility of speeding up CMOS technology 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 circuits 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 best investigated reversible gates. Inexpensive quantum implementations of the Toffoli [2], [11], [17] and the Fredkin [6], [17] gates were found. While the technological costs of Toffoli and Fredkin gates are comparable, it was shown [5] that using both types of gates for the synthesis results in the smaller circuit specifications.

Therefore, a synthesis procedure exploiting both types of gates is of an interest. In this paper we concentrate on the synthesis of the 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. This results in a high degree of interdependence among 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 outputs called garbage (in this paper we consider the following definition of the 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 suggested in [15] for Toffoli network synthesis. 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 introduce bidirectional modification of the basic algorithm and then apply a set of template transforms that reduce the size of the circuit. In this paper we describe and classify the templates used for such transformations in detail.

II. DEFINITIONS

We consider cascades of 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 \( \text{TOF}(C; T) \), where \( C = \{x_{i1}, x_{i2}, \ldots, x_{ik}\} \), \( T = \{x_j\} \) and \( C \cap T = \emptyset \). It maps the Boolean pattern \( (x_1^+, x_2^+, \ldots, x_n^+) \) to \((x_1^{+\prime}, x_2^{+\prime}, \ldots, x_{j-1}^{+\prime}, x_j^{+\prime} \oplus x_{i1}^+, x_{i2}^+, \ldots, x_{ik}^+, x_{j+1}^+, \ldots, x_n^+)\).

Definition 2: For the set of domain variables \( \{x_1, x_2, \ldots, x_n\} \) a generalized Fredkin gate has the form \( \text{FRE}(C; T) \), where \( C = \{x_{i1}, x_{i2}, \ldots, x_{ik}\} \), \( T = \{x_j, x_l\} \) and \( C \cap T = \emptyset \). It maps the Boolean pattern \( (x_1^+, x_2^+, \ldots, x_n^+) \) to \((x_1^{+\prime}, x_2^{+\prime}, \ldots, x_{j-1}^{+\prime}, x_j^{+\prime}, x_{j+1}^{+\prime}, \ldots, x_{l-1}^{+\prime}, x_l^{+\prime}, x_{l+1}^{+\prime}, \ldots, x_n^+)\) iff \( x_{i1}^+, x_{i2}^+, \ldots, x_{ik}^+ = 1 \), otherwise the pattern is unchanged. In other words, the generalized Fredkin gate interchanges bits \( x_j \) and \( x_l \) if, and only if, corresponding product 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 \( TOF(C; x_2) \) is shown in Fig. 1a and the pictorial representation of gate \( FRE(C; x_1, x_2) \) is shown in Fig. 1b.

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 together. 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 \( TOF(S; B) \) gate and for \( |B| = 2 \) is \( FRE(S; B) \). Such a way of writing the gates is needed when we consider a general gate from the Fredkin-Toffoli family and do not want to specify which gate it is. So, if the size of the set \( B \) is not specified, it can be either 1 or 2. The gate shown in Fig. 1c is \( G(C; B) \) where the set \( B \) is not specified.

### III. The Algorithm

The basic algorithm assumes that the function to be synthesized is given as a truth table, and starts synthesis with an empty circuit which realizes the identity function. At every step of the synthesis algorithm, we add some gates from the Fredkin-Toffoli family to the end of the circuit. In the basic algorithm we transform the output specification to the form of the input assigning the gates at the end of the cascade only (thus, building the network in the reverse order).

**Step 0. Idea:** use NOT gates so that they transform the first output pattern to form of the first input pattern.

The first input pattern in the truth table is \((0, 0, \ldots, 0)\) (assume Boolean \( n \)-tuples are arranged in lexicographical order), let the corresponding output pattern be \((b_1, b_2, \ldots, b_n)\).

To bring it to the form \((0, 0, \ldots, 0)\), use gates \( TOF(0; x_i) \) for every \( i : b_i \neq 0 \). After adding the gates to the cascade, update the output part of the table so that the pattern \((b_1, b_2, \ldots, b_n)\) is transformed to the desired form \((0, 0, \ldots, 0)\).

**Step S**, \( 1 \leq S \leq 2^n - 2 \). **Idea:** 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 a real technology) to bring the output pattern to the form of the corresponding input pattern.

The input pattern \((a_1, a_2, \ldots, a_n)\) is the binary representation of natural number \( S \). The pattern in the output part of the same string in truth table is a pattern \((b_1, b_2, \ldots, b_n)\) of higher order. If the order is the same, the patterns are equal, and no action is required. The order of \((b_1, b_2, \ldots, b_n)\) cannot be less than the order of \((a_1, a_2, \ldots, a_n)\) since all such patterns were put to their places during the previous steps of the algorithm.

For pattern \((b_1, b_2, \ldots, b_n)\) to be transformed to the form \((a_1, a_2, \ldots, a_n)\), note that each application of a Toffoli gate is capable of flipping one bit of pattern \((b_1, b_2, \ldots, b_n)\) either from value 0 to value 1 or vice versa, and each Fredkin gate is capable of permuting 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 \((b_1, b_2, \ldots, b_n) \supset (a_1, a_2, \ldots, a_n)\) 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 problem’s 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 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 most significant binary digit of \((b_1, b_2, \ldots, b_n)\) equal one, is greater than the most significant one digit of \((a_1, a_2, \ldots, a_n)\). Therefore, it is taken 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 unit digits of \((a_1, a_2, \ldots, a_n)\).

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 defined as a minimal subset of its unit 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 necessary Toffoli gates. All the controls can be found using the procedure described in the above case.

**Step 2^n - 1.** When the first \(2^n - 1\) patterns are properly aligned, the last pattern will automatically be correct.

**Bidirectional modification.** The basic algorithm works from the output to input by adding gates in one direction starting from the end of the desired cascade. We now describe what happens if the gates are applied in the beginning of the cascade, thus allowing simultaneous synthesis from either side, called the bidirectional modification.

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

**Fredkin gate application.** Apply a Fredkin gate...
application of a template of size $m$ does not change the output of the network. Each symbol, they are all substituted with EXOR. If the assignment was EXOR, then the box is substituted with the EXOR symbol. If the line with the box assigned EXOR contains other box symbols, they are all substituted with EXOR. If the assignment was SWAP, the line with the box becomes the two lines, where the SWAP symbol is put. 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. Further, if a box symbol in a circuit is not specified, it can be either EXOR or SWAP which are substituted into the circuit by the above rules. Now we proceed with the classification.

**IV. TEMPLATE SIMPLIFICATION TOOL**

Let a **size $m$ template** be a sequence of $m$ gates (a circuit) that realizes the identity function. The template size $m$ must be independent of the smaller size templates, e.g. for a given template size $m$ no application of any set of templates of smaller size can decrease the number of gates. For a template $G_0 \ G_1 \ ... \ G_{m-1}$ its **application** is one of the two operations: 

**Forward application.** A sequence of gates in the network which matches the sequence $G_i \ G_{i+1} \ \text{mod} \ m \ ...	ext{mod} \ m \ G_{i+k} \ \text{mod} \ m$ in the template $G_0 \ G_1 \ ... \ G_{m-1}$ is replaced with the sequence $G_{i-1} \ G_{i-2} \ \text{mod} \ m \ G_{i-3} \ \text{mod} \ m \ ...	ext{mod} \ m \ G_{i-k} \ \text{mod} \ m$, where $k \in M$, $k \geq \frac{m}{2}$.

**Backward application.** A sequence of gates in the network which matches the sequence $G_i \ G_{i+1} \ \text{mod} \ m \ ...	ext{mod} \ m \ G_{i+k} \ \text{mod} \ m$ is replaced with the sequence $G_{i-1} \ G_{i-2} \ \text{mod} \ m \ G_{i-3} \ \text{mod} \ m \ ...	ext{mod} \ m \ G_{i-k} \ \text{mod} \ m$, where $k \in M$, $k \geq \frac{m}{2}$.

It can be shown that each of the described template applications does not change the output of the network. Each application of a template of size $m$ for parameter $k > \frac{m}{2}$ leads to a reduction in the number of gates. Further, to reduce the number of different templates, a classification appears to be useful.

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

- according to the number of elements in $B_i$, $G_i$ is written as $\text{TOF}$ (for $|B_i| = 1$) or $\text{FRE}$ (for $|B_i| = 2$);
- $S_i$ is written as a union of sets (C) and single variables $(t)$: $S_i = C_{i1} + C_{i2} + ... + C_{ik} + t_{i1} + t_{i2} + ... + t_{ik}$;
- if $|B_i| = 1$, it is written as a single variable, $t_j$; if $|B_i| = 2$ it is written as the union $t_j + t_k$;
- all the sets are disjoint: $C_{ij} \cap C_{kj} = C_{ij} \cap t_k = t_k \cap t_l = \emptyset$.

In order to classify the templates, we need to discuss the box notation in more detail. If the box is found in a network, there are certain rules for changing the network when the operation of EXOR or SWAP is assigned to the box. If the assignment was EXOR, then the box is assigned with the EXOR symbol. If the line with the box assigned EXOR contains other box symbols, they are all substituted with EXOR. If the assignment was SWAP, the line with the box becomes the two lines, where the SWAP symbol is put. 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.

**m=1.** There are no templates of size 1, since every gate changes at least two input patterns.

**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 two gates 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: $\text{TOF}(C_1, t_1) \ TOF(C_1, t_1)$ and $\text{FRE}(C_1, B_1) \ \text{FRE}(C_1, B_1)$ $(B_1 = t_1 + t_2)$ as shown in Fig. 2.

**m=3.** There are no templates of size 3.

**m=4.** There are several classes of size 4 templates. A very important class is the **passing rule** (analogy of the passing rule from [15]). It defines when the order of the two gates in a network can be changed. All the cases are shown in Fig. 3 with the line marking done for the disjoint notation.

Next class is **semi passing rule**, Fig. 4. Its main feature is equity of the first and third gates, and the fourth gate being the second gate transformed by the target of the first gate. Finally, a class that we call **Fredkin definition** (Fig. 5), being a generalization of the popular circuit simulating Fredkin gate with the Toffoli gates [17].

**m=5.** There is only one class of templates of size 5 (Fig. 5). Although this is a large class, and one would expect to see less applications, since it is harder to match large than to match smaller classes, in practice this class is the most useful.
of Toffoli networks (NCT) us to say that our algorithm produces near optimal circuits which on average are 105.9% of the optimal size. This allows since the methods use different bases for synthesis. Further, of a three variable reversible function.

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

### Table I

<table>
<thead>
<tr>
<th>Size</th>
<th>NCT</th>
<th>Agrawal-Jha</th>
<th>NCTSF</th>
<th>We</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>12</td>
<td>12</td>
<td>18</td>
<td>18</td>
</tr>
<tr>
<td>2</td>
<td>102</td>
<td>102</td>
<td>184</td>
<td>184</td>
</tr>
<tr>
<td>3</td>
<td>625</td>
<td>625</td>
<td>1318</td>
<td>1290</td>
</tr>
<tr>
<td>4</td>
<td>2780</td>
<td>2642</td>
<td>6474</td>
<td>5680</td>
</tr>
<tr>
<td>5</td>
<td>8921</td>
<td>7503</td>
<td>17695</td>
<td>13209</td>
</tr>
<tr>
<td>6</td>
<td>17049</td>
<td>13620</td>
<td>14134</td>
<td>13914</td>
</tr>
<tr>
<td>7</td>
<td>6817</td>
<td>12488</td>
<td>496</td>
<td>5503</td>
</tr>
<tr>
<td>8</td>
<td>32</td>
<td>3297</td>
<td>0</td>
<td>512</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>30</td>
<td>0</td>
<td>9</td>
</tr>
<tr>
<td>WA:</td>
<td>5.866</td>
<td>6.100</td>
<td>5.134</td>
<td>5.437</td>
</tr>
</tbody>
</table>

m=6. We found one class of templates of size 6 (Fig. 5).

### V. Results

We wrote a program that synthesizes functions using the algorithm and then, applies the template tool as a primary circuit simplification procedure. We ran our program exhaustively for all reversible functions with 3 variables and compared the results of our algorithm to the results of optimal synthesis. Table I shows how many functions with 3 inputs can be realized with \( k = 0.9 \) gates in optimal synthesis with the model gates NOT, CNOT, and Toffoli (calculated in [18]), heuristic synthesis with NOT, CNOT, and Toffoli (calculated in [1]), optimal synthesis with the model gates NOT, CNOT, Toffoli, SWAP and Fredkin, and for the presented algorithm realization. Circuits in column We were synthesized in {FIXME} seconds. WA shows the weighted average of the circuit size of a three variable reversible function.

It is hard to compare our synthesis results to those of [1], since the methods use different bases for synthesis. Further, it can be observed that our algorithm produces the circuits which on average are 105.9% of the optimal size. This allows us to say that our algorithm produces near optimal circuits for reversible functions of a small number of variables. Also note, that the heuristic synthesis of Fredkin-Toffoli networks results in a better weighted average than the optimal synthesis of Toffoli networks (NCT column).

### A. Benchmarks

The algorithm does not 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 an Athlon XP 2400+ computer with 512 Mb of RAM, it took approximately 25 minutes to synthesize the network for the largest specification, cycle3_17. Runtimes for smaller specifications are noticeably smaller, e.g., 8.9 seconds for the second largest function that has size 15. For initially irreversible functions, we created a reversible specification using methods discussed in [12], [14].

Our synthesis 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 actual circuits that we reported in this paper can be found online [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” stays for Toffoli gates only, “f” - for Fredkin gates only, and “ft” when the circuit contains both Fredkin and Toffoli gates. It is interesting to notice that Fredkin gates may not help the synthesis (ham15), but sometimes help to reduce the size of the network significantly (hwb5). Also, sometimes when Fredkin gates are allowed, the synthesized circuit may not contain Fredkin gates and be smaller than a circuit synthesized with the condition that Fredkin gates are not allowed (5mod5).

Three of the functions for which we reported the circuits were used as benchmarks in [1]. It is possible to make a fair comparison of the results, since our circuits for these functions use Toffoli gates only. Both we and [1] synthesized function add3 with 4 gates (further, this 4-gate realization is believed to be optimal). Both we and [1] synthesized function rd53 with 12 gates. However, the circuit presented in [1] is incorrect. Finally, we synthesized cycle2_10 with 19 gates, while [1] presented a circuit with 26 gates.

### VI. Conclusion

In this paper, we presented a two-step algorithm for synthesis of the reversible circuits using the most investigated reversible gates, Toffoli and Fredkin. Our algorithm uses the minimal amount of garbage. It consists of two parts. First, given a truth table specification of a function, we synthesize a circuit using the unidirectional or bidirectional algorithm with the gate control size reduction. Once a circuit is created, the template tool is applied for its simplification. The templates are introduced and classified.
Finally, we wrote a program to test our synthesis approach. First, we exhaustively synthesized all size 3 reversible functions and compared the size of the synthesized circuits to the optimal. It turned out that on average our synthesis method produced the circuits that are only 105.9% off the optimal size. Lastly, we tested our software on some benchmark functions and achieved good results.

REFERENCES