Toffoli Gate Cascade Generation Using ESOP Minimization and QMDD-based Swapping

J. E. Rice
Dept. of Mathematics & Computer Science
University of Lethbridge
Lethbridge, AB, Canada

K. B. Fazel & M. A. Thornton
Dept. of Computer Science and Engineering
Southern Methodist University
Dallas, TX, USA

K. B. Kent
Faculty of Computer Science
University of New Brunswick
Fredericton, NB, Canada

Abstract

Two methods for Toffoli gate cascade synthesis of reversible logic circuits are presented. One is based on previous work [3], utilizing an ESOP minimization technique and then applying template-matching [10]. The other is based on a QMDD representation of a Toffoli cascade and determining an ordering that implements the desired function. Experimental results are presented showing the feasibility of both techniques.

1. Introduction

Reversible computing is becoming a more and more popular area of research. One reason for this is that power consumption is becoming a large factor in many devices, while another is the growing interest in quantum computing. Bennett [2] showed that for a circuit to not dissipate power due to information loss, it must be reversible. Some researchers such as Frank [4] state that a computer based on reversible logic operations should be able to reuse a fraction of the signal energies that comes close to 100%, a result first identified in [7]. In addition to this desirable property is the known relationship between reversible and quantum computing; since all quantum gates are reversible, reversible computing can be considered a special case of quantum computing [14]. Many researchers believe that logic synthesis for classical reversible circuits is a first step towards synthesis of quantum circuits [17].

The classic synthesis problem involves translating a specification or description of a function into some description of a circuit implementation. For reversible logic there are many suggested solutions to this in the literature, including [3, 10, 5] and [6]. The problem for many of these techniques lies in the problem of how to represent the function to be synthesized, either initially or during processing. Most of them require an input specification whose size is exponential in the number of variables.

This paper presents two approaches to reversible logic synthesis, both resulting in a cascade of generalized Toffoli gates. The first approach (Method 1) takes two previously published techniques and applies them in sequence. The first of these, introduced in [3], synthesizes a function represented as an exclusive-or sum-of-products (ESOP) to a cascade of reversible Toffoli gates. This technique is one of few in the literature that bases its starting representation and intermediate manipulation on a list of cubes (or products) rather than a truth table, and also allows for the initial circuit specification to be irreversible. We then post-process the output of this technique with the template-matching technique proposed in [11] and refined in [10]. As noted in [3] the initial ESOP-based technique shows a clear trade-off between synthesis speed and quality of the results. That is, while the algorithm performs very quickly, resulting circuits often have more gates than the known optimal solutions. Template-matching [10] is a technique that can take a non-optimal reversible specification and optimize it through the use of matching portions of the reversible cascade with known-optimal sub-circuits, referred to as templates. The goal of this work is to determine if such a technique can produce closer to optimal circuits in a reasonable amount of time.

The second approach (Method 2) uses row moves to sort rows of a given $0, 1$ permutation matrix until the result is the identity. Such a permutation matrix is one way to represent

a reversible logic function. The motivation behind this approach is that there is a great deal of research in Computer Science on how to do sorting, thus it seems reasonable to think that reducing the difficult problem of reversible logic synthesis to a sorting problem can shed some light on how best to approach our problem. We show that for a given pair of rows in the permutation matrix a responding reversible logic gate cascade can be generated that moves row i to row j. Based on this we can then apply sorting algorithms to reversible logic synthesis problems. Since the size of the permutation matrix is exponential in the number of dependent variables, we implement the row swapping operations as graph operations over the Quantum Multiple-Valued Decision Diagram (QMDD) data structure [12]. The use of QMDDs allows for most relatively large reversible circuit specifications to be represented in a compact manner, and as an added bonus the row-swapping operation is implemented in a very efficient manner using the QMDD software.

The paper progresses as follows: Section 2 provides some background on reversible logic in general in preparation for the detailed discussions of the various techniques in Sections 3 and 4. Section 5 gives an overview of related work. Experimental results are presented in Section 6, followed by a discussion of these results.

2. Background

In this field when one refers to a reversible function we are referring to a function that is bijective. For example, Figure 1 gives two functions, one of which (A) is reversible while the other (B) is not. A logic gate can then be considered reversible if the function it computes is bijective [17], and a circuit is considered reversible if it consists entirely of reversible gates.

$x\ y$	$x'\ y'$
00	00
01	10
10	01
11	11

(A)

$x\ y$	$f(x,y)$
00	0
01	0
10	0
11	1

(B)

Figure 1. (A) An example of a reversible function. (B) An example of a non-reversible function.

While many reversible gates have been defined, this work uses only generalized Toffoli gates, of which the NOT gate is a special case. The symbols for a variety of Toffoli (TOF) gates are shown in Figure 2. The NOT gate has the usual behaviour; that is $(x) \rightarrow (x \oplus 1)$ where $\oplus$ denotes the exclusive-or operator. This can be considered to be a TOF1 gate, or a Toffoli gate with one input. A Toffoli

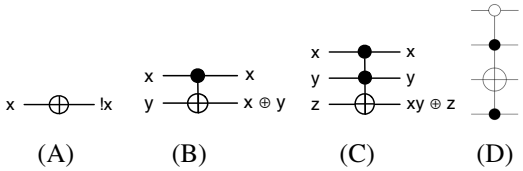

Figure 2. (A) NOT gate, or TOF1 gate. (B) TOF2 gate. (C) TOF3 gate. (D) The generalized Toffoli gate $T(0,1,2,1)$.

gate with more than one input uses the first $n-1$ inputs as control inputs. For example, a TOF2 gate has behaviour $(x, y) \rightarrow (x, x \oplus y)$ where x is considered a control input as it controls the effect of the gate on the input y. This can be extended to n inputs such that TOFn has the behaviour $(x_1, x_2, \ldots, x_n) \rightarrow (x_1, x_2, \ldots, x_{n-1}, x_1 x_2 \cdots x_{n-1} \oplus x_n)$.

For the second method presented in this paper we allow a generalized form of Toffoli gates in that a control line may be asserted positively or negatively. To denote such a Toffoli configuration we use the notation $T(i_1, i_2, \ldots, i_n)$ where i_j denotes the jth line in the Toffoli gate. The behavior of each line is denoted as follows:

- -1 is unused,
- 0 denotes a negative control line,
- 1 denotes a positive control line, and
- 2 denotes the target line.

In Figure 2(D), the control lines with a white dot are negatively asserted while the black dots denote positively asserted control lines.

3. Method 1: ESOP-based Toffoli Gate Cascade Generation with Template Matching

In this section we describe the ESOP-based method for generating a Toffoli gate cascade. This is the first step in a two-part process, the second of which is applying template-matching, described below in Section 3.3.

3.1. ESOPs

An exclusive-or sum-of-products (ESOP) representation is a variation on the more traditional sum-of-products (SOP) representation. Both are two-level representations used in traditional logic synthesis. For example the function $f(x, y, z) = xy + yz$ is shown in a SOP form. If we replace the OR (+) operator with exclusive-or ($\oplus$) then the function becomes an ESOP, for example $g(x, y, z) = xy \oplus yz$. The reader is directed to [16] for more information.

3.2. ESOP-based Generation of Toffoli Gate Cascades

Because the exclusive-or operator is, by its nature, a reversible operator, a function thus expressed as an ESOP can easily be converted to a reversible cascade of gates. The procedure used in [3] is described as follows. We assume a circuit is given in an ESOP cube-list representation. The circuit to be generated will require $2n + m$ signals where n is the number of inputs to the function and m is the number of outputs. In the best case for a truly reversible function $m = n$, but in practice many of the benchmarks used are non-reversible functions with extra (garbage) outputs added to allow for a reversible implementation. More discussion of this can be found in Section 6.3. The $2n$ signals are required in order to represent each input in both its regular and complemented form. The algorithm then generates a Toffoli gate for each cube of each output in the list of ESOP cubes. The basic algorithm is outlined in Figure 3. An example

```
basicCascadeGen(esop)
  cascade.toffoliList = empty;
  //create signals
  foreach i in esop.inputs
    cascade.addQubit(i, positive);
    cascade.addQubit(i, negative);
  foreach o in esop.outputs
    //add a constant 0 qubit for each output
    cascade.addQubit(o, constant 0)
  //create TOF gates
  foreach c in esop.cubes
    foreach o in esop.outputs
      If c in onset(o)
        //add a toffoli gate
        t = new ToffoliGate
        t.target = cascade.getQubit(output)
        foreach literal in c
          t.addControl(cascade.getQubit(literal))
        cascade.addToffoli(t)
```

Figure 3. Algorithm for generating a Toffoli cascade from an ESOP representation.

of an ESOP cube list and the resulting Toffoli cascade are shown in Figure 4. A number of optimization techniques have been included in this method, most notably to reduce the number of input signals required. Details of these are given in [3].

3.3. Template Matching

The basic concept of reversible logic synthesis via template-matching was introduced in [11]. In Miller, Maslov and Dueck's original synthesis method they found that the resulting circuits produced gate sequences that could be replaced with shorter sequences that gave the same result. These substitutions evolved into a series of templates consisting of the sequence of gates to be replaced and the

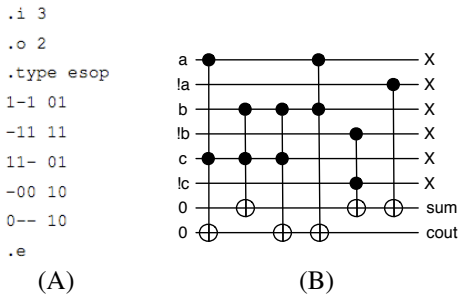

Figure 4. (A) ESOP cube list for a full adder. (B) Result of ESOP mapping method as applied to the cube list in (A).

shorter sequences of gates to replace them with. An example template is shown in Figure 5.

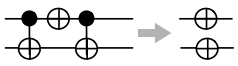

Figure 5. Template 2.2 as given in [11].

4. Method 2: Sorting Based Toffoli Cascade Synthesis

In this section we describe the sorting method for generating a Toffoli gate cascade.

One way to represent a reversible function is as a permutation of the rows of its truth table. For instance, Figure 6 shows a reversible function with the input rows of the truth table labeled on the left and the resulting outputs labeled on the right.

	$x\ y$	$x'\ y'$	
0	00	00	0
1	01	01	1
2	10	11	3
3	11	10	2

Figure 6. A reversible function with the permutation vector $(0, 1, 3, 2)$.

A permutation matrix is a matrix generated by taking a $m \times m$ identity matrix and permuting the rows according to some permutation vector. Since a permutation vector is one way of representing a reversible function of size n then a $m \times m$ permutation matrix (where $m = 2^n$) may be created by reordering the $m \times m$ identity matrix according to the

ordering in the permutation vector p that represents a given reversible function.

Since it is possible to construct a reversible gate cascade that has the effect of moving row i to row j, the problem of constructing such a cascade becomes that of determining the order in which rows should be moved in order to re-sort the permutation matrix into the identity.

In general the algorithm for transforming the permutation matrix M into the identity is

```
while (M!=I)
  (1) in M determine a row i to move to row j
  (2) determine Toffoli operations T
      that perform the row move
```

Work has not progressed a great deal on investigations for step (1), although this is clearly an area where there are a great deal of options. For step (2) we can accomplish a swap of two rows that are adjacent in terms of the bit patterns of their location with only one TOF(n) gate. For instance, suppose we wish to move row i to row j. To determine if a single TOF(n) gate can accomplish this, we can use the value $k = i \oplus j$. If k is a power of two then there exists a generalized TOF(n) gate that will swap the two rows. To determine the configuration of the gate one formulates the Toffoli configuration with the binary representation of i then sets the $(\log k)$th position as the target line. For example, let us assume we have a function with 4 variables for which we wish to swap rows 10 and 14. Then the binary expansions for i and j are 1010 and 1110 respectively, and $k = 0100 = 4$. We set the TOF(4) configuration to initially be $T(1, 0, 1, 0)$, and then set the value at position $\log k = 2$ to 2 indicating the target. This results in a gate configuration of $T(1, 2, 1, 0)$[1].

If k is not a power of two then multiple row moves are necessary, which will correspond to multiple Toffoli gates in the resulting cascade. In this case we propose the use of a Gray code sequence that transforms i to j. For any sequential pair of rows g_a and g_b in the Gray code sequence, $k = g_a \oplus g_b$ is a power of two, and $g_0 = i$ while $g_q = j$, where g_0 is the first row in the Gray code sequence and g_q is the last row. This allows the formulation of a corresponding Toffoli operation, and then the concatenation of the Toffoli swaps one gets from applying the process to each sequential pair of rows in the Gray code sequence corresponds to a row move that moves row i to row j. For instance, in order to move row 0 to row 7 and vice versa we would generate a Gray code to transform the bit pattern 000 into the bit pattern 111, and for each pair in the bit pattern formulate the appropriate TOF(3) gate. An illustration of this process is shown in Figure 7.

We note that the Gray code method only allows for Toffoli gates with a maximum number of control lines. Also, the process described above does not necessarily swap rows,

[1]Recall that the positions are ordered in reverse, *i.e.* $T(i_4, i_3, i_2, i_1)$.

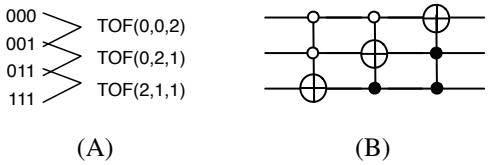

Figure 7. (A) A Gray code sequence for converting 000 into 111, and the corresponding TOF(3) configurations for each pair of rows. (B) The Toffoli gate cascade corresponding to the three TOF(3) configurations in (A).

but instead will move one row to the desired position, possibly affecting other rows in the process. One way to ensure that already sorted rows are not moved would be to ensure that the the ith row is always ordered before the $ith+1$ row. Our proposed algorithm is similar to the Miller, Maslov and Dueck (MMD) algorithm [11] in that rows are ordered such that no previous row move is affected by subsequent row moves. To start, we find the row that contains a 1 in the zeroth column. We denote this as row k. We then move row k to row zero. In general we find the row that contains a 1 in the ith column and swap this row with the ith row. We will refer to this algorithm as the Ordered Row Sorting algorithm. Figure 8 illustrates how the Ordered Row algorithm sorts rows. In this figure the boxes represent 1s while the blank spaces represent 0s, providing a pictorial representation of the permutation matrix being sorted to the identity.

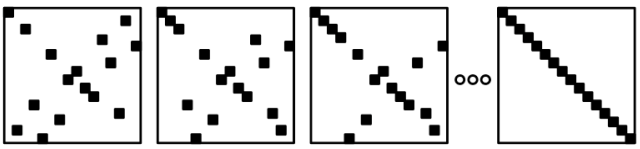

Figure 8. An illustration of the processing carried out by the Ordered Row algorithm.

Another sorting strategy takes advantage of the Quantum Multiple-Valued Decision Diagram (QMDD) data structure [12] and its ability to recursively divide a matrix into quadrants. The QMDD data structure is much like a BDD in that it reuses isomorphic subtrees to minimize memory usage. Figure 9 shows an example QMDD. In the QMDD approach an arbitrary matrix M is first sorted such that we attain the form

$$\begin{bmatrix} A & 0 \\ 0 & B \end{bmatrix}$$

If $A = B$ we then only need to sort A. While we sort A we can choose Toffoli gates that will not only sort A, but

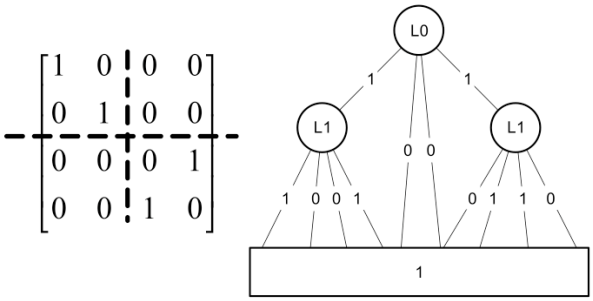

Figure 9. The QMDD structure.

sort B at the same time. We can also recursively sort each sub block similarly. Figure 10 illustrates this approach to sorting the permutation matrix rows.

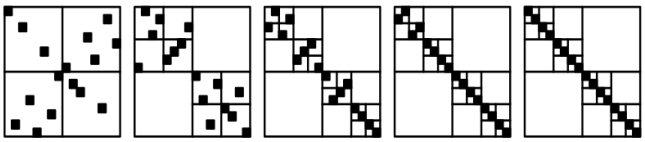

Figure 10. An illustration of the processing carried out by the QMDD sorting algorithm.

5 Related Work

Due to the recent interest in reversible logic there are a wide variety of synthesis techniques in the literature. Here we briefly mention techniques that are related to the approaches detailed in this paper.

5.1 ESOP-related work

The ESOP-based method proposed in [3] and built upon in this paper is similar to techniques suggested by Perkowski *et al.* in [6] and an earlier work [13]. The more recent of these works requires a factorization of each of the ESOPs representing the multiple outputs, and a new class of reversible gates is introduced, allowing modification of two qubits but requiring a significantly higher level of complexity. This method reported achieving good results in terms of gate numbers; for instance, in many cases they required only one gate per product term in the ESOP representation. However the technique still requires the use of some garbage lines, and as the authors themselves state, "[our] cascaded realization of multi-output ESOP generates a large number of garbage outputs and requires a large number of input constants...". Garbage lines are also an issue with the work presented here, and will be further discussed in Section 6.3.

5.2 Sorting-based approaches

As indicated in the prior section, there is some similarity between our suggested sorting method and the well-known method referred to as MMD [11]. The primary difference is in the process of reorganizing the permutation vector and thus generating a Toffoli cascade. The technique used in MMD is quite different from our Gray code technique. It should also be noted that the MMD technique is much more mature, and numerous refinements such as output permutation, a bidirectional option, and control input reduction have been added to this work. Our proposal is still in the early stages of investigation and so refinements of this sort have yet to be added.

6. Experimental Results

In this section we give some experimental results for each of the methods proposed.

6.1. Method 1

The purpose of these experiments is to determine if a post-processing step, such as template-matching, could be used to improve the quality of the reversible circuit as generated by the ESOP-based synthesis process. Initial experiments were designed to follow-up on the results given in [3], which used a common set of irreversible functions that were made reversible by adding garbage lines to the outputs. Unfortunately it soon became clear that the template-matching program could not handle the numbers of variables in many of these benchmarks. Thus the experiments reported here are based on benchmarks whose total number of variables (inputs plus outputs) are 31 or fewer. The process involved running the ESOP-based synthesis with varying parameters. The resulting reversible circuit from each set of parameters were then template-matched using the program supplied by D. M. Miller. The template-matching step was attempted twice; once minimizing for gate count and once minimizing for quantum cost. These results use only a small subset of 14 templates; future work will attempt to incorporate the complete set of 220 templates. Although full data on the experiments, including processing time, was recorded, we do not report on the speed of the various algorithms here since the initial experiments demonstrated quite clearly that the ESOP-based synthesis technique is very fast, and can complete synthesis of very large circuits (for instance, those with more than 200 variables) in very reasonable processing times [3]. The authors are not aware, at this time, of any other reversible synthesis technique that can compete with this approach in terms of speed and the ability to process large functions.

6.1.1 Gate count minimization

Table 5 shows, for each benchmark tested, the smallest circuit size achieved in terms of Toffoli gates. We found that when using the template-matching to minimize for gate count, of the 71 benchmarks, 17 returned the same best result. In 26 cases the ESOP-based technique had the best results, while in 28 cases the template-matching resulted in the best results. In most cases there was no spectacular improvement. The best improvement achieved was a reduction from 38 to 22 gates, but this was not the usual case.

An interesting phenomenon is illustrated in Table 1. This table shows how the parameter α, as used by the ESOP-based technique, can result in the same size of circuit, but clearly has an effect on the subsequent template-matching step.

filename	num signals	α	ESOP tof gates	template tof gates
f51m.pla	22	0	151	156
f51m.pla	22	.25, .5, .75	151	174
f51m.pla	22	1	151	185
apex4.pla	28	0	5079	4950
apex4.pla	28	.25, .5, .75	5079	4993
apex4.pla	28	0	5079	5011

Table 1. Table showing how differing parameter values for the first step can affect the template-matching second step.

α is used as a cost metric to determine how early on in processing variables must be considered (see [3] for details).

6.1.2 Quantum cost minimization

The quantum cost is computed using the algorithm given in the template-matching code supplied by Miller. This algorithm is based on quantum cost values for Toffoli gates as given in [8]. The quantum cost of a gate is determined by how many elementary quantum gates are needed to implement the reversible gate in question. In some cases fewer elementary gates may be used if there are garbage lines available for use. The reader is directed to [8] and [1] for further details.

In this case the template-matching technique always results in a circuit with either the same or lower quantum cost than does the initial ESOP-based processing. In fact, the average improvement is 38%. Complete results are shown in Table 6. Again it is worth highlighting some unexpected results such as are shown in Table 2. In some situations, for instance for benchmark `x2.pla` the quantum cost after the initial ESOP-based step is lowest for $\alpha = 0$. However, the lowest cost after template-matching is for $\alpha = .25, .5, .75$, which initially (after ESOP processing) had the highest quantum cost. This raises the question of what the initial circuits looked like structurally, and how these structural changes either helped or hindered the template-matching process.

filename	num signals	α	ESOP qc	template qc
x2.pla	17	0	595	426
x2.pla	17	.25, .5, .75	600	377
x2.pla	17	1	601	430

Table 2. Table showing how an increase is quantum cost in the initial step can result in a lowered quantum cost after template-matching.

6.2. Method 2

The Ordered Row and QMDD algorithms were both implemented in C++ utilizing the QMDD data structure. Additionally the two sorting approaches were compared against the most basic form of the MMD algorithm (no bidirectionality, input reduction or output permutation). We note that both of our sorting algorithms allow for the use of the generalized Toffoli gates, whereas the MMD algorithm ensures only positively asserted control lines are generated. To allow for a fair comparison the cascades of generalized Toffoli gates from our techniques are expanded to include NOT gates. Results are shown in Table 3. Columns 1 and 2 denote the circuit under investigation and the number of inputs. Column 3 shows results for the Ordered Row algorithm, where the number of Toffoli gates found by the algorithm is given as the first value, and the number of NOT gates needed to remove negatively asserted lines is shown as the second value. Column 4 shows similar results for the QMDD approach. The final column shows results from MMD.

One can see the large disparity between the MMD cascade sizes and Ordered Row sorting generated cascade sizes. The large number of gates in the row swap approaches are due to the large number of row moves that are necessary when only moving a single row at a time. A variety of approaches to mitigate this problem are suggested in Section 6.3.

6.3. Summary & Discussion

The link between these two approaches lies in our identification of a key problem in existing reversible logic synthesis techniques: the reliance on function representations

Circuit	signals	Ordered TOF/NOT	QMDD TOF/NOT	MMD
2of5-b	6	10/60	10/60	18
2of5-c	6	28/130	16/124	17
2of5-dueck	6	30/146	39/212	20
Dimitri-3-new	3	10/14	9/18	8
Dimitri-3	3	6/12	8/16	6
Dimitri-4	4	20/42	18/62	22
Hayes-1	1	4/8	3/6	1
and2	2	1/0	1/0	1
cycle-b	3	9/10	6/16	3
cycle	3	15/18	6/8	3
fredkin	3	3/0	3/6	3
gray3	3	4/4	3/4	3
gray4	4	12/22	7/16	6
hard-14-a-r	3	12/8	9/24	10
hard-14-a	3	6/8	8/18	14
hard-14-b	3	8/14	10/28	9
miller3	3	7/4	5/16	5
miller4	4	15/18	7/30	7
nasty-10	3	8/16	8/22	12
nasty-11	3	8/16	8/22	12
nasty-swap	3	6/8	8/18	6
rm-2001	4	20/40	26/86	10
selector	6	5/20	5/20	11
test	3	6/8	10/22	6

Table 3. Sort-based Toffoli synthesis results.

that are exponential in size. Both of these approaches represent an attempt to avoid this problem. A brief summary of the advantages and disadvantages of each of these proposals is given in Table 4. This table clearly indicates areas where future work must be targeted: reductions of additional garbage lines for both techniques, and in how to deal with non-reversible functions for the sorting-based technique. Although the sorting-based methods did not fare very well against the MMD algorithm we believe it is possible to formulate the MMD algorithm into a sorting framework. This gives hope for the sort-based methods in the sense that if a smarter heuristic is used to choose Toffoli gates then smaller overall circuits can be attained. Furthermore, this could allow for much larger circuits to be synthesized using the MMD technique since the QMDD data structure would be used instead of explicit truth table representations. The current Gray code approach provides a minimum number of rows to be moved, assuming only single-row moves are permitted. We believe that forcing a lot of rows to move at any given time is a better heuristic than only allowing a few rows to move; in other words, it would be better to try to find Toffoli gates that do not require a maximum number of control lines, and thus have lower quantum cost. It should be possible to use the Gray code as a basis for finding Toffoli cascades that allow for many rows to move. Additionally, the number of NOT gates can be reduced if we take into account the fact that neighboring NOT gates may override one another.

	ESOP	Sorting
non-reversible input	yes	no
non-exponential input	yes	no
always results in reversible cascade	yes	yes
garbage lines	yes	no
internal representations	cube-list	QMDD

Table 4. A comparison of the two proposed reversible logic synthesis techniques.

Similarly, for the ESOP-based technique some reduction of the garbage requirements is needed to make this technique competitive. Maslov and Dueck [9] provide a theoretical minimum for the required number of garbage outputs, and future work (as suggested in Section 7) must find some way to approach this value using the ESOP technique.

7. Conclusions and Future Work

Two techniques for Toffoli gate cascade generation are presented. These are very different techniques in that one is a more mature approach and has been refined to work well on a large number of (irreversible) benchmarks, while the second is in the early stages of development and is thus far restricted to small, reversible benchmarks. Because of this further comparisons between the two are difficult at this stage in the research.

Method 1 consists of a combination of two previously published approaches [3, 10]. When comparing the gate counts of circuits resulting from the ESOP technique to those achieved by preprocessing the results using template-matching, the ESOP technique is quite comparable. Additional tests were run using the template-matching to minimize for quantum cost. In this case the second step of template-matching then improves the results by almost 40%. It is worth noting that the ESOP-based technique was capable of processing extremely large benchmarks, and that for this work the size of benchmarks was restricted to 31 signals in order for the template-matching step to complete in reasonable time. Additionally, this method could be employed as the initial phase of a synthesis method that is used to perform mapping of large and/or irreversible specifications to reversible cascades suitable for optimization using other approaches.

There are many possibilities for future work with this technique, including optimizing the ESOP technique to use templates as the circuit is built. Also, the fact that the initial cascade has the property of allowing interchange of the

position of any two arbitrary gates means that the template-matching approach can be modified to take advantage of this new degree of freedom, since in the MMD method the interchanges of gate positions is limited. Further work is needed to identify why some values of α lead to improved template-matching results, and determine if it is possible to identify ahead of time which functions may lead to substantially better results when template-matching is used so as to best leverage the extra computation time. In addition to these avenues of research, some solution to the garbage problem must be found. Options in this area include post-processing to "fold" garbage lines into other signal lines, or some modification to the basic processing algorithm that allows reuse of input signal lines for outputs. Finally, new work has indicated that the autocorrelation coefficients may be of use in determining variable costs (α), and may lead to an improved cost metric [15].

Method 2 consists of two sorting-based methods for Toffoli cascade synthesis. We propose a technique for determining a Toffoli cascade that will move one row of a permutation matrix. The repeated application of row moves can be used to generate a Toffoli cascade for a given reversible specification. Although the provided sort-based approaches did not compare particularly well against a known method (MMD [11]) we believe that with some modifications, including modified heuristics and combining adjacent NOT gates, this method can eventually provide comparable results. It may be interesting to investigate whether the generated cascades provide good starting points for post-processing methods such as template-matching.

Future work includes determining better sorting algorithms using our row move technique, and also considering the possibility of incorporating Fredkin gates into the sorting approach. Preliminary research has already demonstrated the feasibility of this particular avenue of research.

References

[1] A. Barenco, C. H. Bennett, R. Cleve, D. P. DiVinchenzo, N. M. Argolus, P. Shor, T. Sleator, J. A. Smolin, and H. Weinfurter. Elementary gates for quantum computations. *Physical Review A*, 52(5):3457–3467, Nov 1995.

[2] C. H. Bennett. Logical reversibility of computation. *IBM Journal of Research and Development*, 6:525–532, 1973.

[3] K. Fazel, M. Thornton, and J. E. Rice. ESOP-based Toffoli gate cascade generation. In *Proceedings of the IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM)*, pages 206–209, 2007. Aug. 22–24 2007, Victoria, BC, Canada, IEEE Press.

[4] M. P. Frank. Introduction to reversible computing: Motivation, progress, and challenges. In *Proceedings of the 2nd Conference on Computing Frontiers*, pages 385–390, 2005. May 4–6, Ischia, Italy, ACM Press.

[5] P. Gupta, A. Agrawal, and N. K. Jha. An algorithm for synthesis of reversible logic circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 25(11):2317–2330, Nov. 2006.

[6] M. H. A. Khan and M. A. Perkowski. Multi-output ESOP synthesis with cascades of new reversible gate family. In *Proceedings of the 6th International Symposium on Representations and Methodology of Future Computing Technology (RM 2003)*, pages 144–153, 2003.

[7] R. Landauer. Irreversibility and heat generation in the computing process. *IBM Journal of Research and Development*, 5:183–191, 1961.

[8] D. Maslov and G. W. Dueck. Improved quantum cost for n-bit Toffoli gates. *Electronics Letters*, 39(25):1790–1791, Dec 2003.

[9] D. Maslov and G. W. Dueck. Reversible cascades with minimal garbage. *IEEE Trans. on Computer-Aided Design*, 23(11):1497–1509, Nov. 2004.

[10] D. Maslov, G. W. Dueck, and D. M. Miller. Techniques for the synthesis of reversible Toffoli networks. *ACM Trans. Des. Autom. Electron. Syst.*, 12(4):42, 2007.

[11] D. M. Miller, D. Maslov, and G. W. Dueck. A transformation based algorithm for reversible logic synthesis. In *Proceedings of the 40th Conference on Design Automation (DAC'03)*, pages 318–323, New York, NY, USA, 2003. ACM.

[12] D. M. Miller and M. A. Thornton. QMDD: A decision diagram structure for reversible and quantum circuits. In *Proceedings of the IEEE International Symposium on Multiple-Valued Logic (ISMVL)*, page no. 30 on Proceedings CDROM, 2006. May 17–20.

[13] A. Mishchenko and M. Perkowski. Logic synthesis of reversible wave cascades. In *Proceedings of the International Workshop on Logic Synthesis*, pages 197–202, 2002. June 4-7, New Orleans, USA, IWLS Workshop.

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

[15] J. E. Rice. The autocorrelation transform and its application to the classification of boolean functions. submitted to PACRIM 2009.

[16] T. Sasao. *Switching Theory for Logic Synthesis*. Kluwer Academic Publishers, 1999.

[17] V. V. Shende, A. K. Prasad, I. L. Markov, and J. P. Hayes. Reversible logic circuit synthesis. In *IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, pages 353–360, 2002. November 10–14, San Jose, CA, USA, ACM.

filename	signals	α	esop	template	filename	signals	α	esop	template
ex1.pla	6	-	5	5	inc.pla	16	1	71	66
ex2.pla	6	0	7	7	mlp4.pla	16	0	93	95
ex3.pla	6	-	4	4	5xp1.pla	17	0.25	61	60
majority.pla	6	-	5	5	parity.pla	17	-	16	16
xor5.pla	6	-	5	5	ryy6.pla	17	-	19	19
C17.pla	7	-	8	8	t481.pla	17	-	12	12
cm82a.pla	8	0	16	15	x2.pla	17	-	20	21
f2.pla	8	0.25	14	13	alu3.pla	18	1	60	60
rd53.pla	8	-	20	20	dk27.pla	18	-	15	15
con1.pla	9	0	12	13	sqr6.pla	18	0.25	65	63
9sym.pla	10	0	42	56	add6.pla	19	0	180	180
life.pla	10	-	46	60	alu1.pla	20	1	15	16
max46.pla	10	0	41	53	ex1010.pla	20	0.25	1314	1269
rd73.pla	10	0	55	57	C7552.pla	21	-	89	77
sqn.pla	10	0	45	45	decod.pla	21	-	89	77
dc1.pla	11	0	33	29	dk17.pla	21	0	34	29
sym10.pla	11	0.25	51	66	pcler8.pla	21	-	13	13
wim.pla	11	1	17	14	alu4.pla	22	0	156	187
z4.pla	11	0	36	37	apla.pla	22	1	63	52
cm152a.pla	12	-	8	11	cm150a.pla	22	-	17	24
rd84.pla	12	0	68	72	f51m.pla	22	0	151	156
sqrt8.pla	12	0	27	28	mux.pla	22	-	16	23
adr4.pla	13	0	41	40	tial.pla	22	0	150	174
dist.pla	13	0	124	129	cordic.pla	25	-	5	5
radd.pla	13	0	46	40	cu.pla	25	0	24	17
root.pla	13	1	63	62	gary.pla	26	0	111	104
squar5.pla	13	0.25	36	35	in0.pla	26	0	111	104
clip.pla	14	0	98	101	apex4.pla	28	0	5079	4950
cm42a.pla	14	0	38	20	cm151a.pla	28	-	26	24
cm85a.pla	14	0	48	49	misex3.pla	28	0.25	262	263
pm1.pla	14	0	38	20	misex3c.pla	28	0	281	285
sao2.pla	14	0	23	22	table3.pla	28	0.25	88	77
dc2.pla	15	0	53	53	cm163a.pla	29	0	26	22
misex1.pla	15	-	43	42	in2.pla	29	0.25	88	85
alu2.pla	16	0	71	84	frg1.pla	31	0	26	27
example2.pla	16	0	71	84					

Table 5. Table showing gate counts for the initial ESOP technique and subsequent template matching. Only the smallest resulting circuits (when minimizing for gate count) are listed.

filename	signals	α	e cost	t cost	filename	signals	α	e cost	t cost
ex1.pla	6	-	7	7	inc.pla	16	0	2132	801
ex2.pla	6	0.25	153	132	mlp4.pla	16	1	3827	2278
ex3.pla	6	-	97	92	5xp1.pla	17	0.25	1349	722
majority.pla	6	-	147	120	parity.pla	17	-	32	32
xor5.pla	6	-	7	7	ryy6.pla	17	-	4892	4892
C17.pla	7	-	97	83	t481.pla	17	-	237	237
cm82a.pla	8	0.25	167	96	x2.pla	17	0.25	600	377
f2.pla	8	1	274	152	alu3.pla	18	0.25	2653	1997
rd53.pla	8	1	289	196	dk27.pla	18	1	252	190
con1.pla	9	0	207	183	sqr6.pla	18	0	1090	703
9sym.pla	10	0	5781	4633	add6.pla	19	1	6362	3812
life.pla	10	1	4074	3002	alu1.pla	20	1	243	243
max46.pla	10	0	4432	3470	cmb.pla	20	-	910	688
rd73.pla	10	0	1105	835	ex1010.pla	20	0	183726	53428
sqn.pla	10	0	2170	1331	C7552.pla	21	-	1924	325
dc1.pla	11	0.25	539	169	decod.pla	21	-	1924	325
sym10.pla	11	0.25	9717	7423	dk17.pla	21	0	1976	1044
wim.pla	11	0	281	153	pcler8.pla	21	0.25	343	267
z4.pla	11	0	674	454	alu4.pla	22	0	48778	37760
cm152a.pla	12	-	219	219	apla.pla	22	0.25	4051	1577
rd84.pla	12	1	2598	1897	cm150a.pla	22	-	844	844
sqrt8.pla	12	0	584	352	f51m.pla	22	0	34244	24995
adr4.pla	13	0.25	770	536	mux.pla	22	-	826	826
dist.pla	13	0.25	7414	3453	tial.pla	22	0	49510	38895
radd.pla	13	1	798	478	cordic.pla	25	0	349522	100837
root.pla	13	0	3486	1586	cu.pla	25	0	1332	756
squar5.pla	13	0	476	306	gary.pla	26	1	22196	7556
clip.pla	14	0.25	6616	3034	in0.pla	26	1	22196	7556
cm42a.pla	14	0	582	142	apex4.pla	28	0	256857	35033
cm85a.pla	14	0	2228	906	cm151a.pla	28	0	966	404
pm1.pla	14	0	582	142	misex3.pla	28	0	122557	46268
sao2.pla	14	0.25	7893	3652	misex3c.pla	28	0	118578	45785
co14.pla	15	-	3508	1764	table3.pla	28	0	86173	17099
dc2.pla	15	0.25	1956	1037	cm163a.pla	29	0	988	452
misex1.pla	15	0	1017	339	in2.pla	29	0	24388	7877
alu2.pla	16	0	5215	4346	frg1.pla	31	0	15528	15497
example2.pla	16	0	5215	4346					

Table 6. Table showing quantum costs for the initial ESOP technique and subsequent template matching. Only the smallest resulting circuits (when minimizing for quantum cost) are listed.