Technical Report: Three techniques for generation of Toffoli gate cascade implementaions of reversible circuits

TR-CSJR1-2009
J. E. Rice
University of Lethbridge
j.rice@uleth.ca

1 Introduction

Reversible logic is becoming a "hot" topic of research for a variety of reasons, one of which is its connections to quantum computing. Since prior reports have elaborated on this, this report will simply direct the reader to these [20] and to introductory works reversible logic such as [4, 32] and [1]. for more details on the motivation behind this area.

This report provides the reader with background in the current status of synthesis for reversible logic, paying particular attention to three approaches:

- ESOP-based Toffoli-gate cascade synthesis,
- template-matching, and
- sorting-based Toffoli-gate cascade synthesis.

2 Background

2.1 Reversible Logic

[26] provides the following definitions:

Definition 2.1 *a gate is reversible if the (Boolean) function it computes is bijective,*

and

Definition 2.2 *a well-formed reversible logic circuit is an acylic combinational logic circuit in which all gates are reversible, and are interconnected without fanout.*

This definition assumes that the circuit is strictly combinational; considerations for sequential logic are addressed in [19]. In general, a function is reversible if there is a one-to-one and on-to mapping from the inputs to the outputs (and vice versa) of the function. For example, the function shown in Figure 1(A) is reversible, while the function shown in Figure 1(B) is not.

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

(A)

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

(B)

Figure 1: (A) An example of a reversible function. It is one-to-one and on-to. (B) An example of a non-reversible function.

2.1.1 Representations

A key consideration in this area lies in the question of how to represent a reversible function and/or an implementation of a reversible function as a reversible circuit.

One way to represent a reversible function is as permutation of the rows of its truth table. For instance, Figure 2 shows a reversible function with the input rows of the truth table labeled on the left and the resulting outputs labeled on the right. A permutation vector for this function could therefore be given as $(0, 1, 3, 2)$.

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

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

Other techniques treat reversible functions as special classes of traditional logic functions, and then use traditional representations such as the espresso PLA format [23], which is a tabular form of the well-known sum-of-products (SOP) representation, or the exclusive-or version of this known as a ESOP. An ESOP form results when the OR (+) operator in a SOP expression is replaced with the exclusive-or ($\oplus$) operator. The reader is directed to [27] and [24] for more information on the ESOP representation. Both the use of SOPs and ESOPs have the benefits of not being exponential in the number of variables, and of being able to leverage the extensive amount of previous work (see, for example, [24]) in minimization of these formats.

More recently two standard formats for representing reversible logic functions and circuits have been proposed: the RevLib (rl) format and the The RevLib format can be used either to specify a reversible function, before an implementation has been determined, or to define a circuit implementing the reversible function. The function specification consists of all the truth table outputs for the function. Input are not given, as it is assumed that the outputs are given in standard row ordering (inputs ordered 0 to $2^n - 1$). Circuit specifications consist of a list of reversible gates that, when cascaded in the order given, will implement the function. Details of this proposed format are given in [34].

Maslov, Dueck and Scott also propose a format for representation of reversible logic circuit implementations, again in terms of a list of reversible gates. Details of this proposed format are given in [11].

Most of the commonly used representations of reversible logic functions are exponential in the number of signals, or variable used by the function. Some authors have suggested the use of decision diagrams to mitigate this problem. The reader is directed to a variety of works for more information in this area: [14, 28, 17, 5, 29, 30].

2.1.2 Reversible Gates

Because a reversible gate must be bijective, this means that the traditional inverter is considered reversible, while the traditional AND gate is not.

x	$\overline{x}$
0	1
1	0

(A)

x y	$x \cdot y$
0 0	0
0 1	0
1 0	0
1 1	1

(B)

Table 1: (A) The traditional inverter, a reversible gate. (B) The traditional AND gate, which is irreversible.

The most commonly used reversible gates include the NOT gate (or the inverter), a SWAP gate, and variations on these. Table 2 lists the behaviour of some reversible gates. The Feynman and Toffoli are

gate	behaviour
NOT	$(x) \rightarrow (x \oplus 1)$
Feynman	$(x, y) \rightarrow (x, x \oplus y)$
Toffoli	$(x, y, z) \rightarrow (x, y, xy \oplus z)$
SWAP	$(x, y) \rightarrow (y, x)$
Fredkin	$(x, y, z) \rightarrow (x, z, y)$ *iff* $x = 1$

Table 2: The behaviour of a selection of more commonly used reversible logic gates.

in essence extensions of the NOT gate, while the Fredkin extends the behaviour of the SWAP gate. It is common to extend these behaviours to any number of control lines. In this case the Toffoli gate may be referred to as TOFn, where n is the total number of signal lines the gate is operating on. A TOFn gate may also be referred to as a $(n, n-1)$ Toffoli gate, where n is the total number of lines and $n-1$ is the number of control lines. This is somewhat redundant, however, and so we will generally use the first notation in this report.

The generalized behaviour of the TOFn gate can be characterized as

$$(x_{n-1}, x_{n-2}, \ldots, x_1, x_0) \rightarrow (x_{n-1}, x_{n-2}, \ldots, x_1, x_0 \oplus c)$$

where $c = x_{n-1} \cdot x_{n-2} \cdot \cdots \cdot x_1$. The variables used in the computation of c are referred to as control lines, while x_0 in this case is the target. Any of the variables may be designated as the target, in which case that variable is not used in the computation of c.

A further generalization allows the control variables to be used in either their positive or complemented form. A notation for this assigns a numeric value to the behaviour of each line as given in Table 3. For

-1	unused
0	negative control
1	positive control
2	target

Table 3: The mapping of values to functionality for each line in a generalized Toffoli gate.

example, a TOF4 gate that has x_3 as a negative control line, x_2 as a positive control line, x_1 as the target and x_0 as a positive control line would be denoted TOF(0,1,2,1) (or alternatively $T(0, 1, 2, 1)$).

Figure 3 illustrates the symbols used for the gates discussed in this report.

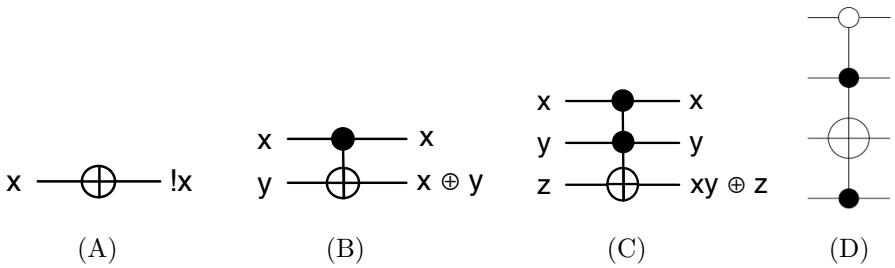


Figure 3: (A) NOT gate, or TOF1 gate. (B) TOF2 gate. (C) TOF3 gate. (D) The generalized Toffoli gate $T(0, 1, 2, 1)$.

2.2 Terminology

In most cases this report will explain terminology as it is needed. In addition we will try to point out when differing terminology is used for the same or similar concepts by different authors. However a short comment on one point is useful: although we refer to "inputs" and "outputs" of a reversible function, this is a misnomer. This is because a reversible function may be implemented using some quantum technology, where wires and traditional technologies cannot be used, and where inputs/outputs are really starting/ending states of some technological entity implementing the function.

Given this we will assume that the reader understands this caveat and in most cases we refer to either lines of the circuit, assuming that these will be implemented by some reversible technology equivalent, or to the variables in the function to be implemented.

3 Existing Reversible Synthesis Techniques

As recent researchers have found, synthesizing reversible logic functions is not a trivial process. In fact, early researchers in this area generally limited themselves to trials on functions with only 3 inputs. This report focuses on three techniques. We first introduce the basic idea behind each of the techniques.

3.1 Maslov *et al.'s* Template Technique

Maslov, Miller and Dueck have published a number of papers leading to what I will refer to here as the template technique. Parts of this algorithm are often referred to as MMD. Details for this technique are first introduced in [13] and further elaborated on in [9] and [10].

The technique begins with a truth table describing a reversible function, and produces a cascade of Fredkin and Toffoli gates. The steps are as follows:

1. use NOT gates to transform the first row of outputs in the truth table to $00\cdots0$, and then update all output rows according to the changes.
2. moving down the truth table use the simplest possible (*i.e.* having the fewest control lines) to bring each output pattern to the form of its corresponding input pattern without influencing previous rows in the table.
3. apply the template simplification rule

For example, given the function shown in Table 4 the tranformation process is as follows:

xyz	$x'y'z'$
000	000
001	011
010	100
011	101
100	110
101	010
110	111
111	001

Table 4: A $n = 3$ reversible function to synthesize.

xyz	$x'y'z'$
000	000
001	001
010	100
011	111
100	110
101	010
110	101
111	011

a)

xyz	$x'y'z'$
000	000
001	001
010	010
011	111
100	110
101	100
110	011
111	101

b)

xyz	$x'y'z'$
000	000
001	001
010	010
011	011
100	110
101	100
110	111
111	101

c)

xyz	$x'y'z'$
000	000
001	001
010	010
011	011
100	100
101	110
110	101
111	111

d)

Table 5: Truth tables corresponding to the steps in the template-based transformation synthesis approach.

Step 0 No change is required for this step; row 0 is already correct

Step 1 The goal is now to transform the output pattern 011 to match the input pattern 001; one way to do this is to apply the function $y' = y \oplus z$, which matches the Feynman gate, and then apply this function to all of the outputs resulting in the truth table shown in Table 5a).

Step 2 Row 2 has 100 in the output column and 010 in the input column; swap x' and y' to make them match. The resulting truth table is shown in Table 5b).

Step 3 The next change required is in row 3 wherew there is 111 in the output column and 011 in the input column. The Toffoli gate can be applied to make the required change: $x' = x \oplus yz$. The result is shown in Table 5c).

Step 4 In row 4 the goal is to change 110 to 100; apply $y' = y \oplus x$. The result is shown in Table 5d).

Now all output patterns match the inputs, and the resulting cascade of gates is shown in Figure 4. The final step is to examine the cascade of gates and replace any series of gates that match a variety of templates determined by the authors (see [9] for details) if such a replacement reduces the number of gates in the series. Templates have been determined by identifying all series of gates which realize the identity function, and by enumerating exhaustively the templates consisting of 1–6 gates. These templates have then been classified to reduce the number of different templates. These classes are based on the functionality of the templates.

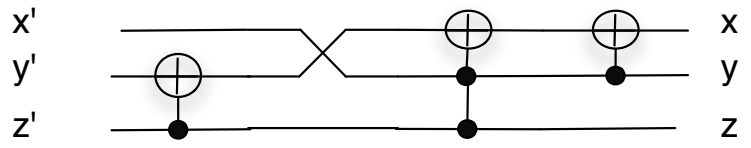


Figure 4: The resulting cascade of gates for the truth table in Table 4.

3.2 ESOP-based Synthesis

[3] introduces a technique based on the ESOP representation. This technique is attractive for two reasons:

1. it requires as input a non-exponential representation of the function to be synthesized, and
2. it is so simple that actually performing the synthesis should prove to be very fast and potentially scalable to larger functions.

The problem with the basic technique is that for a reversible function of n variables the implementation requires $3n$ lines: one line for each input and one for the negated form of each input, as well as one line for each output. The technique consists of creating a circuit with the required number of lines and inserting one gate for each cube in the ESOP cubelist. If we allow CNOT (Toffoli) gates to control more than one output, then such a technique will result in a cascade with exactly p gates, p being the number of cubes (or products) in the ESOP cubelist. An example is shown in Figure 5.

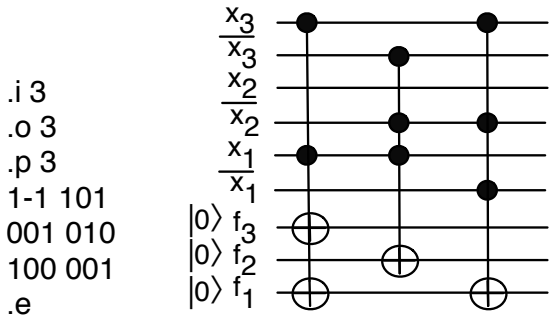


Figure 5: a) An ESOP cubelist. b) The resulting cascade of gates when generating a circuit from the cubelist in a).

Optimizations to address these problesm were introduced in [3]. One optimization consists of sorting the cubes such that all cubes containing the non-negated form of a particular variable are moved to the top of the list, while all cubes containing the negated form of that same variable were moved to the bottom of the list. This process is then repeated within each of the two halves of the list for the next variable, and so on. This allows the removal of lines representing the negated form of each input variable and instead use NOT gates to switch as needed. In the very worst case, where all minterms are included in the cubelist, this would result in $2^n - 1$ inverters, since the 2^n cubes could be ordered in such a way that only one bit changed between each cube. This is extremely unlikely, however. A cost function is used to determine which variables should be sorted on first – for instance, variables that do not use their negated version can appear anywhere in the ordering, while variables that are used extensively AND with a relatively even number of negated and non-negated appearances would benefit from being dealt with early on in the ordering process. In determining such a cost function both of these factors are considered. We assign a value of $+1$ to each place where the variable appears as a 1 in the cubelist; a value of -1 to each place where the variable appears as a 0 in the cubelist, and a value of 0 if the variable appears as a don't care. Then we wish to

maximize $\sum |x_i|$ to determine if a variable appears as a non-don't care value in a majority of the cubes, and to minimize $|\sum x_i|$ to determine if a variable appears fairly often in both its negated and non-negated forms, rather than just in one or the other.

$$C_{x_i} = p_1(\frac{1}{\sum |x_i|}) + p_2(|\sum x_i|) \tag{1}$$

Equation 1 gives a formula with two parameters that can be varied to alter the contribution of each part of the function, and each of the summations will run from cube 1 to cube p in the list. Experimentation has shown that setting the two parameters $p_1 = p_2 = 1$ allows the second portion of the equation, the $|\sum x_i|$ to overwhelm the first portion, and so the parameter p_1 should be set to some value to allow the two contributions to be more equal. It is likely that p_1 should vary with p (the number of cubes).

The reader is directed to [21] for more information on ESOP-based logic synthesis.

3.3 Sorting-based Synthesis

A new method proposed in [22] utilizes row moves to convert a given 0, 1 permutation matrix to the identity. In effect, such row moves may be viewed as sorting rows of the matrix. Since sorting is such a well-known problem in computer science the expectation is that the body of knowledge surrounding this area can be applied to the problem of reversible logic synthesis.

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. Because the sizes of the permutation transfer matrices are exponential in the number of dependent variables, the row swapping operations are implemented as graph operations over the Quantum Multiple-Valued Decision Diagram (QMDD) data structure [14]. This allows for most relatively large reversible circuit specifications to be represented in a compact manner, and the row-swapping operation is implemented in a very efficient manner using QMDD.

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.

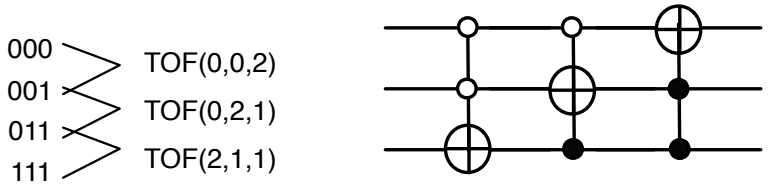
For step (2) we first focus on row moves that require exactly one TOF(n) gate. A swap of two rows that are adjacent in terms of the bit patterns of their location will only require one TOF(n) gate. For instance, suppose we wish to move row i to row j. To determine if a single TOF(n) gate can accomplish this, we can use the value $k = i \oplus j$. If k is a power of 2 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 intially 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].

To formulate a row move that requires multiple row swaps the authors suggest 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, for the situation in Table 6, it is clear that we need to move row 0 to row 7, and vice versa. We would thus 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 6.

x y z	x'y'z'
000	111
001	001
010	010
011	011
100	100
101	101
110	110
111	000

Table 6: A reversible function in which row 0 and row 7 have been swapped.



(A) (B)

Figure 6: (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).

NOTE: minimization of the ESOP is performed by the EXORCISM software, as described in [15] and as available for download [12].

4 Conclusions & Future Work

This report provides examples of three techniques used in reversiblel logic synthesis. We note that there are a variety of other approaches not addressed here; some of these are listed below for the reader to pursue as desired:

- De Vos *et al.* [18], [31], [33], [32]
- Shende *et al.* [25], [26]

[1]Recall that the positions are ordered in reverse, *i.e.* $T(x_3, x_2, x_1, x_0)$.

- synthesis for Peres gates [2]
- Perkowski *et al.* [6], [16], [8], [7]
- Wille *et al.* [34], [36],[35]

In particular we highlight the works in [21] and [35]; future work will include ways to incorporate these two techniques.

References

[1] A. N. Al-Rabadi. *Reversible Logic Synthesis*. Springer-Verlag, 2004.

[2] James Donald and Niraj K. Jha. Reversible logic synthesis with Fredkin and Peres gates. *Journal of Emerging Technologies in Computer Systems*, 4(1):2:1–2:19, April 2008.

[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. Kerntopf. A New Heuristic Algorithm for Reversible Logic Synthesis. In *Proceedings of the Design Automation Conference (DAC)*, pages 834–837, 2004. June 7–11, San Diego, CA, USA, ACM.

[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] M. Lukac, M. Kameyama, M. Perkowski, and P. Kerntopf. Decomposition of reversible logic function based on cube-reordering. In *Proceedings of the Reed-Muller 2011 Workshop*, pages 63–70, 2011. May 25–26, Tuusula, Finland.

[8] M. Lukac, M. Perkowski, H. Goi, M. Pivtoraiko, C. H. Yu, K. Chung, H. Jee, B-G. Kim, and Y-D. Kim. Evolutionary approach to quantum and reversible circuits synthesis. *Artificial Intelligence Review, Special Issue on Artificial Intelligence in Logic Design*, 20(3–4):361–417, December 2003.

[9] D. Maslov, G. W. Dueck, and D. M. Miller. Synthesis of fredkin-toffoli reversible networks. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 13(6):765–769, June 2005.

[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. Maslov, G. W. Dueck, and N. Scott. Reversible logic synthesis benchmarks page, 2008. http://webhome.cs.uvic.ca/ dmaslov.

[12] A. Michshenko. Exorcism software. http://web.cecs.pdx.edu/ alanmi/research/min/minEsop.htm.

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

[14] 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.

[15] A. Mishchenko and M. Perkowski. Fast heuristic minimization of exclusive sum-of-products. In *Proceedings of the 5th International Reed-Muller Workshop*, pages 242–250, 2001. August 2001, Starkville, Mississippi.

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

[17] M. Perkowski, L. Joziwak, A. Mixhchenko, A. Al-rabadi, A. Coppola, A. Buller, X. Song, M. Khan, S. Yanushkevich, V. P. Shmerko, and M. Chrzanowska-Jeske. A general decomposition for reversible logic. In *Proceedings of the International Workshop on Methods and Representations (RM)*, pages 119–138, 2001. August 10–11, Starkville, Mississippi, USA, RM Workshop.

[18] Y. Van Rentergem, A. De Vos, and K. De Keyser. Using group theory in reversible computing. In *Proceedings of the 2006 IEEE Congress on Evolutionary Computation (CEC2006)*, pages 2397–2404, 2006.

[19] J. E. Rice. An introduction to reversible latches. *The Computer Journal*, 51(6):700–709, 2007.

[20] J. E. Rice. TR-CSJR2-2007: Considerations for determining a classification scheme for reversible boolean functions. Technical report, University of Lethbridge, 2007. Available at http://www.cs.uleth.ca/ rice/publications.html.

[21] J. E. Rice. TR-CSJR1-2012: Esop-based methods for toffoli gate cascade generation. Technical report, University of Lethbridge, 2012.

[22] J. E. Rice, K. B. Fazel, M. A. Thornton, and K. B. Kent. Toffoli gate cascade generation using ESOP minimization and QMDD-based swapping. In *Proceedings of the International Symposium on Representations and Methodology of Future Computing Technologies (RM2009)*, 2009. May 23–24, Naha, Okinawa, Japan.

[23] R. Rudell. Espresso minimization tool man pages.

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

[25] 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.

[26] V. V. Shende, A. K. Prasad, I. L. Markov, and J. P. Hayes. Synthesis of reversible logic circuits. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 22(6):710–722, June 2003.

[27] B. Steinbach and A. Mishchenko. SNF: a special normal form for ESOPs. In *Proceedings of the International Symposium on Representations and Methodology of Future Computing (RM2001)*, pages 66–81, 2001.

[28] M. Thornton, D. M. Miller, and D. Goodman. A decision diagram package for reversible and quantum circuit simulation. In *Proceedings of the IEEE Congress on Evolutionary Computation*, pages 8597–8604, July 2006. held at the IEEE World Congress on Computational Intelligence (WCCI).

[29] G. F. Viamontes, I. L. Markov, and J. P. Hayes. Graph-based simulation of quantum computation in the density matrix representation. *Quantum Information & Computation*, 5(2):113–130, 2005. Preprint available at quant-ph/0403114.

[30] G. F. Viamontes, I. L. Markov, and J. P. Hayes. QuIDDPro: High-performance quantum circuit simulation, 2006. http://vlsicad.eecs.umich.edu/Quantum/qp/.

[31] A. De Vos and Y. Van Rentergem. Reversible computing: From mathematical group theory to electronical circuit experiment. In *Proceedings of the 2nd Conference on Computing Frontiers*, pages 35–44, 2005. May 4–6, Ischia, Italy, ACM Press.

[32] A. De Vos and Y. Van Rentergem. Synthesis of reversible logic for nanoelectronic circuits. *International Journal of Circuit Theory and Applications*, 35(3):325–341, 2007. Published online 17 April 2007 in Wiley InterScience (www.interscience.wiley.com).

[33] A. De Vos, Y. Van Rentergem, and K. De Keyser. The decomposition of an arbitrary reversible logic circuit. *Journal of Physics A: Mathematical and General*, 39(18):5015–5035, May 2006.

[34] R. Wille, D. Große, L. Teuber, G. W. Dueck, and R. Drechsler. RevLib: An online resource for reversible functions and reversible circuits. In *International Symposium on Multiple Valued Logic*, pages 220–225, 2008. RevLib is available at http://www.revlib.org.

[35] R. Wille, M. Soeken, and R. Drechsler. Reducing the number of lines in reversible circuits. In *Proceedings of the Design Automation Conference (DAC)*, pages 647–652, 2010. June 13–18, Anaheim, California.

[36] Robert Wille and Rolf Drechsler. Bdd-based synthesis of reversible logic for large functions. In *Proceedings of the 46th Annual Design Automation Conference*, DAC '09, pages 270–275, New York, NY, USA, 2009. ACM.