A Synthesis Method for MVL Reversible Logic*

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

Gerhard W. Dueck
Faculty of Computer Science
University of New Brunswick
Fredericton, NB, Canada E3B 5A3
gdueck@unb.ca

Dmitri Maslov
Department of Computer Science
University of Victoria
Victoria, BC, Canada V8W 3P6
dmaslov@uvic.ca

Abstract

An r-valued m-variable reversible logic function maps each of the r^m input patters to a unique output pattern. The synthesis problem is to realize a reversible function by a cascade of primitive reversible gates.

In this paper, we present a simple heuristic algorithm that exploits the bidirectional synthesis possibility inherent in the reversibility of the specification. The primitive reversible gates considered here are one possible extension of the well-known binary Toffoli gates.

We present exhaustive results for the 9! 2-variable 3-valued reversible functions comparing the results of our algorithm to optimal results found by breadth-first search. The approach can be applied to general m-variable, r-valued reversible specifications. Further, we show how the presented technique can be applied to irreversible specifications. The synthesis of a 3-input, 3-valued adder is given as a specific case.

1. Introduction

A binary or MVL circuit is reversible if it maps each input pattern to a unique output pattern. Landauer [7] proved that traditional binary irreversible gates lead to power dissipation in a circuit regardless of its implementation. Recently, Zhirnov *et al.* [14] calculated that power dissipation in future CMOS (scaled for the year 2016 in accordance with the ITRS plan) leads to impossible heat removal, and thus the impossibility of speeding up CMOS technology devices. Bennett [2] 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 as regards further technological advances. Binary reversible circuits have been studied for their potential application in low-power CMOS design, quantum computation [8][10] and optical computing.

This paper addresses the problem of synthesizing an MVL reversible specification in terms of basic MVL reversible gates. Practical implementation will require a suitable technology for implementation of those gates. Possible implementation in quantum technology is addressed by the authors in []. At present, quantum implementation is limited to a small number of qubits. MVL coding reduces the number of qubits required (lines in the circuit) thus allowing for larger specifications in current technology.

We do not elaborate further on technology issues but do make the following assumptions:

(i) fan-out between gates is not permitted;
(ii) loops are not permitted; and
(iii) permutation of connections between gates is permitted.

We employ MVL reversible gates that are extensions of the binary reversible NOT, Feynman [4] and Toffoli gates [13]. The gates considered are those introduced by De Vos *et al*. [3] and simple extensions of those gates.

The gates considered here are not the only possible MVL reversible gates nor are they the only possible extension to the Toffoli gate. Picton [11][12] introduced an MVL generalization of the binary Fredkin gate. Al-Rabadi [1] has considered a generalization of the Toffoli gate where XOR is replaced by mod-sum. Khan *et al.* [6] have considered several MVL (ternary) reversible gates. The set of gates we use here is of interest because of its relative simplicity and consistency and because preliminary investigations indicate that implementation should be relatively efficient. The reader will observe that due to the simple nature of our synthesis method it can be extended to other sets of primitive reversible gates.

The methods and examples re presented for simplicity presented for the ternary case. However, it will be clear from the simplicity of the synthesis approach presented that it is straightforward to apply it to higher-radix logic.

Synthesis of reversible logic is quite different from conventional synthesis. Since loops are not permitted, a

* This work was supported in part by research grants from the Natural Sciences and Engineering Research Council of Canada.

reversible logic circuit can be specified as a simple sequence of gates. Further, since fan-out is not permitted, and assuming an appropriate technology, a reversible logic circuit can realize the inverse specification simply by applying the gates in the reverse order. Hence, synthesis can be carried out from the inputs toward the outputs or from the outputs toward the inputs. The method presented here synthesizes the circuit by working in both directions simultaneously. In addition, it is advantageous to synthesize a circuit for a given specification and also for its inverse taking the solution to be the simpler result. The synthesis method presented here is based on the binary method developed by the authors in [9] but there are certain novel issues to deal with in the MVL case.

The background on reversible logic necessary for this paper is outlined in Section 2. The interested reader is referred to [10] for extensive background. The MVL reversible gates used in this work are described in Section 3. Section 4 presents our synthesis algorithm. An example illustrating the operation of the algorithm is worked in some detail in Section 5. Experimental results are given in Section 6 and an irreversible example, the 3-valued full adder, is shown in Section 7. The paper concludes with observations and ideas for further research in Section 8.

2. Background

Definition 2.1 An m-input, m-output, (written $m \times m$) totally-specified MVL function is reversible if it maps each input assignment to a unique output assignment.

A reversible function thus defines a permutation of the input patterns and there are clearly $r^m!$ r-valued, $m \times m$ reversible functions.

Definition 2.2 An n-input, n-output gate is reversible if it realizes a reversible function.

The synthesis problem is how to realize a given reversible specification using a basic set of reversible gates.

A variety of binary reversible gates have been considered. The common NOT gate realizes a reversible function. Another binary reversible gate is the Fredkin gate [5] which has three inputs and three outputs. The first input is passed through unaltered. The second and third pass through unaltered if the first input is 0 and are exchanged if the first input is 1.

There is also the family of Toffoli gates [13] defined as follows:

Definition 2.3 An $n \times n$ Toffoli gate passes the first n-1 lines (control) through unchanged, and inverts the n^{th} line (target) if the control lines are all 1.

The 2×2 Toffoli gate and the 3×3 Toffoli gate have been named the *controlled-NOT (Feynman)* and *controlled-controlled-NOT* gates, respectively.

3. MVL Reversible Gates

A. De Vos *et al.* [3] have considered the cycle and negation operations in Table 1, denoted C1 and N respectively, and the controlled versions of those gates, denoted CC1 and CN, given in Table 2, as generators of the group of all 2×2 3-valued reversible logic functions. The notation used here is that symbols such as x and y denote the line values on one side of a reversible gate while x^+ and y^+ denote the corresponding values on the other side of the gate.

x	C1 x^+	N x^+
0	1	2
1	2	1
2	0	0

Table 1

x	y	CC1 x^+	CC1 y^+	CN x^+	CN y^+
0	0	0	0	0	0
0	1	0	1	0	1
0	2	0	2	0	2
1	0	1	0	1	0
1	1	1	1	1	1
1	2	1	2	1	2
2	0	2	1	2	2
2	1	2	2	2	1
2	2	2	0	2	0

Table 2

gates	C1-CN	C1 CC1-CN	C1-N CC1-CN	C1-C2-N CC1-CC2-CN
0	1	1	1	1
1	4	6	8	12
2	13	31	52	93
3	39	130	280	597
4	115	498	1,342	3,224
5	326	1,777	5,692	15,042
6	897	5,924	20,992	57,951
7	2,395	18,089	63,292	144,039
8	6,107	47,849	128,159	127,056
9	14,660	99,576	118,635	14,750
10	32,268	126,981	23,516	115
11	62,145	58,192	906	
12	96,237	3,795	5	
13	97,705	31		
14	43,902			
15	5,816			
16	243			
17	7			
Avg.	11.97	9.39	8.11	7.16

Table 3

x	C2 x^+
0	2
1	0
2	1

x	y	CC2 x^+	y^+
0	0	0	0
0	1	0	1
0	2	0	2
1	0	1	0
1	1	1	1
1	2	1	2
2	0	2	2
2	1	2	0
2	2	2	1

Table 4

We have written a program to determine the number of gates for circuits realizing the 9! = 362,880 2×2 3-valued reversible logic functions for different sets of basic reversible gates. The program performs a breadth-first search so the first circuit found for each function uses the minimal number of gates. Results are shown in Table 3.

Cyclic inversion (C1) together with controlled negation (CN) is the only operation pair that can realize all 362,880 functions. As shown, the circuits can be quite long. Adding, controlled cycle (CC1) and negation (N) in turn both improve the results. The latter is the set of operators suggested by De Vos *et al.* [3].

The rightmost column in Table 3 shows the further improvement gained by adding cycle by 2 (C2) and controlled cycle by 2 (CC2) defined in Table 4. Clearly, N and CN are self-inverse. C1 and C2 are the inverse of each other. Hence, CC1 and CC2 are the inverse of each other.

Negation can be extended to any *r*-valued logic as $\overline{x}$ = (*r*-1)-*x*. There are *r*-1 cyclic inversions for *r*-valued logic defined in the obvious way. Controlled cycles and controlled negation can be generalized to the *n*×*n* cases for *r*-valued logic as specified in the following definition.

Definition 3.1 An *n*×*n* *r*-valued controlled unary gate passes the first *n*-1 lines (control) through unchanged, and applies a specified operation to the n^{th} line if the control lines are all *r*-1, otherwise the target line is passed through unaltered.

The Fredkin gate can clearly be extended to the MVL case, but we do not consider that in this paper.

4. A Synthesis Method

An *m*×*m* reversible MVL specification is a mapping

$$F : Q \rightarrow Q$$

where Q is the set of r^m *m*-tuples of *r* values. We only consider the totally-specified case in this paper. As F is reversible, there is of course a corresponding inverse mapping

$$F^{-1} : Q \rightarrow Q$$

x	D x^+	E x^+
0	0	1
1	2	0
2	1	2

Table 5

We will write $F(\mathbf{x}) = \mathbf{y}$ (conversely $F^{-1}(\mathbf{y}) = \mathbf{x}$) where $\mathbf{x}, \mathbf{y} \in Q$ and shall denote the elements of Q as the ordered set $\{q_0, q_1, \ldots, q_{r^m-1}\}$ hence q_i is the *m*-ary *r*-valued expansion of *i*.

Our synthesis method operates by finding a sequence of MVL reversible gates whose effect is to transform F (and of course F^{-1}) to the identity mapping. It is based on the binary synthesis approach we developed in [9].

There are two aspects of our approach which must be noted at this point. First while the discussion so far has assumed control values are always *r*-1 (2 for ternary), our algorithm allows control values to be any value ≠ 0. The reason will become obvious in the description of the algorithm. Extra cycles or negations can always be inserted so that only *r*-1 control values are necessary, but we anticipate that in many implementations of the basic reversible gates, arbitrary control values will be available. This extension leads to the following:

Definition 4.1 A generalized *n*×*n* *r*-valued controlled unary gate passes the first *n*-1 lines (control) through unchanged, and applies a specified operation to the n^{th} line if each of the control lines equals the control value ($0 < v < r$) specified for that line; otherwise the target line is passed through unaltered.

Also, because our algorithm processes the specification in a fixed order, we need an additional operation D (see Table 5) which we assume is also available as a controlled gate.

Definition 4.2 Given a reversible specification *F*, the distance between *F* and the identity is given by

$$\Delta(F) = \sum_{j=0}^{r^m-1} \delta(q_j, F(q_j))$$

where for two *r*-valued *m*-tuples *a* and *b*

$$\delta(a,b) = \sum_{k=0}^{m-1} | a_k - b_k |$$

Synthesis Procedure

Input: reversible specification F_0

Output: an ordered set of gates G_{r^m} that implements the initial reversible specification

1. i = 0 and $G_0 = \phi$
2. if ($F_i(q_i) = q_i$) skip to step 9
3. let S = ordered set of gates to map $F_i(q_i)$ to q_i
4. let T = ordered set of gates to map $F_i^{-1}(q_i)$ to q_i
5. for j = 0 to i set $F_{i+1}(q_j) = q_j$

6. if |S| < |T|
   (a) for j = i+1 to $r^m - 1$ set $F_{i+1}(q_j)$ the result of applying the gates in S (in order) to $F_i(q_j)$
   (b) set G_{i+1} to $G_i \mid < \ldots, S_1, S_0 >$ (note | denotes concatenation)
7. if |S| > |T|
   (a) for j = i+1 to $r^m - 1$ set $F_{i+1}^{-1}(q_j)$ the result of applying the gates in T (in order) to $F_i^{-1}(q_j)$
   (b) set G_{i+1} to $< T_0, T_1, \ldots > \mid G_i$
8. if |S| = |T| we apply either 6(a) and (b) or 7(a) and (b) whichever yields the smallest $\Delta(F_i)$ (in the event of a tie, 6(a) and (b) are applied)
9. $i = i + 1$
10. if ($i < r^m - 1$) go to step 2

The following notes clarify the steps of the algorithm.

1. Initialization step.
2. If $F(q_i) = q_i$ no transformation is required for this pass.
3. The gates to map $F_i(q_i)$ to q_i are chosen to (*a*) minimize the number of gates and to (*b*) ensure that when they are applied they have no affect for any $j < i$. The actual procedure for choosing the gates is given in detail below.
4. This is analogous to step 3 but is being applied in the opposite direction.
5. Due to the gate selection process F_{i+1} is identical to F_i for q_0 to q_i.

6., 7. and 8. S is a set of gates that map an output pattern to match the corresponding input pattern. T is a set of gates that map that input pattern to match the output pattern. The algorithm selects the smaller set and in the case of a tie the set that results in a specification with smallest distance to the identity mapping. Note that gates mapping an output pattern to match the input pattern are appended to the end of G and in reverse order (step 6b). Conversely, gates mapping an input pattern to match the output pattern are appended to the beginning of G in order.

9. and 10. are iteration control.

The key to the synthesis algorithm is the selection of a set of gates to map an m-tuple q_k to another q_i. By construction, $k > i$. Also, the gates must be chosen so that they have no affect on any $q_j, j < i$. Both of these stipulations arise from the fact the algorithm goes through the specification in order. The gate selection subprocedure is given below in terms of a generic reversible specification F which of course can be the inverse of another specification.

Gate Selection Subprocedure

Input: a reversible specification F and an i for which we want to select gates to transform $F(q_i)$ to q_i

Output: an ordered set of gates G performing the required transformation

1. For ease of notation let $a = F(q_i)$ and let $b = q_i$. Set $G = \phi$.
2. $k = 0$
3. if ($a_k = b_k$) skip to step 6
4. Select a gate g that transforms a_k to b_k with control values (if needed) being the smallest number of $a_p, p \neq k$, so that the value of the control set when treated as a single value is $\geq b$. The gate chosen in the event of a choice is the one that yields a transformed specification F^t such that $\Delta(F^t)$ is minimal.
5. set G to $G \mid g$
6. $k = k + 1$
7. if($k < m$) go to step 3

Step 4 is the core of the above procedure. A difference between the binary and the MVL situations is the variety of choice for mapping one value to another. For example, the case for ternary (r = 3) is shown in Table 6. In the event there is a choice in constructing the control set, we choose higher control values.

It is very important to note that in Table 6 we do not use C1 for $1 \rightarrow 2$ or C2 for $2 \rightarrow 1$ as these would always modify an entry earlier in the specification. This is an artifact of our simple algorithm processing the specification in order. We include E in Table 6 for completeness but note that it is not used in our current implementation.

We can further exploit reversibility by applying the algorithm to F and then to F^{-1}. Even though the algorithm itself exploits reversibility, it is heuristic and applying it to both the original and the inverse can produce a different circuit. The circuit for F^{-1} can of course simply be applied in reverse to realize F.

Both the synthesis and gate selection procedures are greedy in that they make choices to optimize the circuit based on only local information. No backtracking or look-ahead is used.

transform	choice
$0 \rightarrow 1$	C1, E
$0 \rightarrow 2$	C2, N
$1 \rightarrow 0$	C2, E
$1 \rightarrow 2$	D
$2 \rightarrow 0$	C1, N
$2 \rightarrow 1$	D

Table 6

5. Example

A trace of a 2-variable, 3-valued example will illustrate the operation of the synthesis algorithm. The initial specification is given in Table 7(a). The choice is to map output pattern 21 to 00 or input pattern 21 to 00. While these seem equivalent, they are not. The synthesis procedure chooses to map the input pattern 21 as this yields the resulting specification with smaller $\Delta(F)$. The gates required are N(x) and C2(y). The resulting specification is given in Table 7(b) with bold type denoting the changes which in this case are to the input.

(a)		(b)		(c)	
xy	x^+y^+	xy	x^+y^+	xy	x^+y^+
00	21	**22**	21	22	2**2**
01	20	**20**	20	20	20
02	22	**21**	22	21	2**1**
10	12	1**2**	12	12	12
11	10	1**0**	10	10	10
12	11	1**1**	11	11	11
20	02	**02**	02	02	02
21	00	**00**	00	00	00
22	01	**01**	01	01	01

Table 7

The entries 00, 01, …, 20 are properly aligned. The next choice is to map 21 to 22 on either the input or the output side. Both require a single gate which is a controlled D applied to y with control $x = 2$. The resulting specification is given in Table 7(c) and is the identity so the process is complete. The circuit is shown in Figure 1.

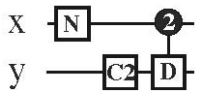


Figure 1

6. Experimental Results

We have implemented the synthesis method described in Section 4 in C. Even on a 750MHz PC with 256MB RAM running Windows XP, the computation time for functions of a few variables is negligible. The exhaustive enumeration described below takes a few minutes.

We have applied our synthesis algorithm to the 9! = 362,880 2×2 3-valued reversible logic functions. The results are shown in Table 8. As noted above, the algorithm allows control values of 1 and 2, and uses D operations. E operations are not used. The Table shows the results for applying the algorithm to F only, and the results for applying the algorithm to F and F^{-1} and taking the shortest circuit. For comparison, we include the optimal results found by breadth-first search when control values of 1 and 2, and D operations are permitted.

gates	Algorithm F Only	Algorithm F and F^{-1}	Optimal
0	1	1	1
1	24	24	24
2	301	315	335
3	2,395	2,593	3,407
4	11,743	12,954	25,255
5	34,755	39,061	114,095
6	72,217	80,699	187,569
7	97,192	103,663	32,173
8	81,978	79,099	21
9	43,886	34,338	
10	14,849	8,612	
11	3,246	1,437	
12	293	84	
Avg.	7.11	6.92	5.60

Table 8

7. Irreversible Example

We present the case of a 3-valued full adder as an example of applying the above reversible logic synthesis technique to an irreversible specification. The full adder has three inputs x_0, x_1, x_2 and two outputs which are the *sum* and *carry* defined in the obvious way (Table 9).

To apply the above methods to this specification we must embed it in a larger reversible specification. Looking at the truth table for the full adder we see that output pattern 10 appears 7 times. Since all the output patterns in the reversible specification must be unique, this requires we add at least two 'garbage' outputs. The term garbage refers to the fact those outputs are not part of the useful output set.

The reversible specification must have the same number of inputs and outputs, so we must add at least one 'constant' input x_3 which we add as the most significant entry in the truth table and which we define so that setting it to 0 yields the correct *sum* and *carry* functions.

The difficult part is how to define the two garbage outputs so that the resulting specification is reversible and to minimize the resulting circuit. We draw upon our experience of the full adder in the binary case [9]. The reversible specification is given by

$$x_0^+ = sum(x_0, x_1, x_2)$$
$$x_1^+ = x_1 \oplus x_2$$
$$x_2^+ = x_2$$
$$x_3^+ = carry(x_0, x_1, x_2) \oplus x_3$$

where ⊕ denotes sum mod-3. Note that the ordering of the outputs is important and that the two garbage outputs are placed between the *sum* and *carry*. The circuit only produces the correct carry when $x_3 = 0$. Choosing the best output ordering is in general difficult. Here it was done by inspection. Our synthesis algorithm is sufficiently fast that for small problems, one could consider all possible output permutations. How to choose an output order for a large problem is an open question.

Applying the synthesis method described above to the specification given by (3) yields the circuit shown in Figure 2. Sixteen gates are required. Each shows the type of gate, the target line and any control lines and values applicable. The algorithm gives 27 gates for the inverse specification.

8. Conclusion

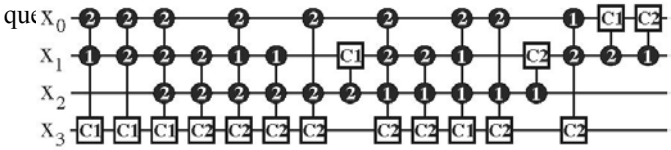
This paper introduces a simple heuristic algorithm for the synthesis of MVL reversible circuits composed of MVL reversible gates based on the ideas of De Vos *et al*. [3] that are one possible generalization of binary Toffoli gates [13]. While the initial results are quite promising there is considerable need and scope for further research.

x_2	x_1	x_0	*carry*	*sum*
0	0	0	0	0
0	0	1	0	1
0	0	2	0	2
0	1	0	0	1
0	1	1	0	2
0	1	2	1	0
0	2	0	0	2
0	2	1	1	0
0	2	2	1	1
1	0	0	0	1
1	0	1	0	2
1	0	2	1	0
1	1	0	0	2
1	1	1	1	0
1	1	2	1	1
1	2	0	1	0
1	2	1	1	1
1	2	2	1	2
2	0	0	0	2
2	0	1	1	0
2	0	2	1	1
2	1	0	1	0
2	1	1	1	1
2	1	2	1	2
2	2	0	1	1
2	2	1	1	2

2	2	2	2	0

Table 9

In earlier binary work [9], we presented a similar heuristic algorithm and then gave a template-based reduction procedure that can significantly reduce the size of the circuit. We are currently working on templates for the MVL case. Also, while we were able to find quite a reasonable circuit for a ternary full adder as a reversible circuit, work is needed to identify a general procedure for embedding an irreversible specification within a larger reversible specification so that the resulting reversible circuit is minimal, or at least near-minimal. This is an open que



C1(x3;x0=2;x1=1)
C1(x3;x0=2;x1=2)
C1(x3;x0=2;x1=2;x2=2)
C2(x3;x1=2;x2=2)
C2(x3;x0=2;x1=1;x2=2)
C2(x3;x1=1;x2=2)
C2(x3;x0=2;x2=2)
C1(x1;x2=2)
C2(x3;x0=2;x1=2;x2=1)
C2(x3;x1=2;x2=1)
C1(x3;x0=2;x1=1;x2=1)
C2(x3;x0=2;x2=1)
C2(x1;x2=1)
C2(x3;x0=1;x1=2)
C1(x0;x1=2)
C2(x0;x1=1)

Figure 2

The basic reversible gates considered here are only one possible generalization of Toffoli gates, and there are many other MVL reversible gates that could be considered. We next plan to consider MVL generalizations of the Fredkin gate [5]. Mod-sum based reversible gates may also be considered.

Application of this work depends on the efficient realization of the basic reversible gates in a suitable technology. Our studies reported in [] show that the C1, C2 and N gates and their controlled counterparts will have quite reasonable quantum logic realizations. The D and E gates are somewhat more expensive. As noted earlier, while the procedure and examples are given for the ternary case, the work is readily extended to higher radices.

Finally, the synthesis algorithm presented here is greedy and heuristic. We are extending the approach to use back-tracking to further improve the quality of the solution. We

are also considering an approach which does not necessarily process the input/output patterns in order.

References

[1] Al-Rabadi, A., "New Multiple-Valued Galois Field Sum-of-Product Cascades and Lattices for Multiple-Valued Quantum Logic Synthesis," *6th International Symposium on Representations and Methodology of Future Computing Technologies*, March 2003, pp. 171-182.

[2] Bennett, C., "Logical Reversibility of Computation," *IBM Jour. of Research and Developmen*t, **17**, 1973, pp. 525-532

[3] De Vos, A., B. Raa and L. Storme, "Generating the Group of Reversible Logic Gates," *J. of Physics A: Mathematical and General*, **35**, 2002, pp. 7063-7078.

[4] Feynman, R., "Quantum Mechanical Computers," *Optics News*,**11**, 1985, pp. 11-20.

[5] Fredkin, E., and T. Toffoli, "Conservative Logic," *International Jour. Theoretical Physics*, 1982, pp. 219-253.

[6] Khan, Mozammel H. A., Marek A. Perkowski and Pavel Kerntopf, "Multi-output Galois Field Sum of Products (GFSOP) Synthesis with New Quantum Cascades," *Proc. International Symposium on Multiple-Valued Logic*, May 2003, pp. 146-153.

[7] Landauer, R., "Irreversibility and Heat Generation in the Computational Process," *IBM Journal of Research and Development*, **5**, 1961, pp. 183-191.

[8] Milburn, Gerard J., *The Feynman Processor*, Perseus Books, 1998.

[9] Miller, D. M., D. Maslov, and G. W. Dueck, "A Transformation Based Algorithm for Reversible Logic Synthesis," *Proc. 2003 Design Automation Conference*, June 2003, pp. 318-323.

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

[11] Picton, P., "A Universal Architecture for Multiple-Valued Reversible Logic," *MVL Journal*, **5**, 2000, pp. 27-37.

[12] Picton, P., "Modified Fredkin Gates in Logic Design," *Microelectronics Journal*, **25**, 1994, pp. 437-441.

[13] Toffoli, T., "Reversible Computing," in *Automata, Languages and Programming*, Springer-Verlag, pp. 632-644, 1980.

[14] Zhirnov, V. V., R. K. Cavin, J. A. Hutchby, and G. I. Bourianoff, "Limits to Binary Logic Switch Scaling – A Gedanken Model," *Proc. of the IEEE*, **91**, no. 11, 2003, pp. 1934-1939.