Reversible Function Synthesis with Minimum Garbage Outputs

Gerhard W. Dueck* and Dmitri Maslov
Faculty of Computer Science
University of New Brunswick
Fredericton, N.B. E3B 5A3 CANADA

Abstract

Reversible functions are of interest in the design of low power CMOS circuits and in quantum computing. We describe a synthesis method for a proposed regular structure of generalized Toffoli gates. Multiple output functions are transformed into reversible functions with a minimum of number garbage outputs. The synthesis algorithm is greedy with one level look ahead. The synthesis results from benchmark functions are encouraging. The current version of the algorithm is limited to functions with no more than 10 inputs.

1 Introduction

Energy loss is an important consideration in digital design. Part of the problem of energy dissipation is related to non-ideality of switches and materials. Higher levels of integration and the use of new fabrication processes have dramatically reduced the heat loss over the last decades. The other part of the problem arises from Landauer's principle [5] for which there is no solution. Landauer's principle states that logic computations that are not reversible, necessarily generate heat $kT * \log 2$ for every bit of information that is lost, where k is Boltzmann's constant and T the temperature. For room temperature T the amount of dissipating heat is small (i.e. $2.9*10^{-21}$ joule), but not negligible. The design that does not result in information loss is called reversible. It naturally takes care of heating generated due to the information loss. This will become an issue as the circuits become smaller.

Most gates used in digital design are not reversible. For example the AND, OR, and XOR gates do not perform reversible operations. Of the commonly used gates, only the NOT gate is reversible. A set of reversible gates is needed to design reversible circuits. Several such gates have been proposed over the past decades. Among them are the *controlled-not* (CNOT) proposed by Feynman [1], Toffoli [12], and Fredkin [2] gates. These gates have been studied in detail. However, good synthesis methods have not emerged. Shende *et al.* [11] suggest a synthesis method that produces a minimal circuit with up to 4 input variables. Iwama *et al.* [3] describe transformation rules for CNOT based circuits. These rules may be of use in a synthesis method. Miller [7] uses spectral techniques to find near optimal circuits. Mishchenko and Perkowski [9] suggest a regular structure of reversible wave cascades and show that such a structure would require no more cascades than product terms in an ESOP realization of the function. In fact, one would expect that a better method can be found. The algorithm sketched in [9] has not been implemented. A regular symmetric structure has been proposed by Perkowski *et al.* [10] to realize symmetric functions.

In reversible logic feedbacks and fan-outs are not permitted. This makes the synthesis substantially different. Traditional design methods use, among other criteria, the number of gates as complexity measure (sometimes taken with some specific weights reflecting area of the gate). From the point of view of reversible logic we have one more factor, which may be more important than the number of gates used, namely the number of garbage outputs. Since reversible design methods use reversible gates, where number of inputs is equal to the number of outputs, the total number of outputs of such a network will be equal to the number of inputs. The existing methods ([9]) use analogy of copying gates to keep information on the input of the network, therefore introducing the constant inputs and garbage outputs—information that we do not need for the computation. In some cases garbage is unavoidable. For example, a single out-

*Research supported by the NSERC (CANADA).

put function of n variables will require at least $n-1$ garbage outputs, since the reversibility necessitates an equal number of outputs and inputs.

Previously, we [6] suggested a structure for reversible design that requires minimal number of garbage outputs, therefore solving the "garbage" part of the synthesis problem. In this work we take the proposed structure and design a regular method for the function synthesis. The results of the synthesis method, applied to some benchmark functions are analyzed and compared to the results of [9] and [7].

2 Synthesis of Reversible Functions

2.1 Definitions

In this section we propose a method for reversible logic function synthesis and expand it to multiple output functions in the next section. To make a multiple a output function reversible we may have to add garbage outputs and/or constant inputs. For instance, 2-input 2-output output function $(x, y) \rightarrow (x, xy)$ is not reversible, since the input values cannot be recovered if the output is $(0, 0)$. The $(0, 0)$ output could be the result of the input being one of $(0, 0)$ or $(0, 1)$. One way to make this function reversible, is to consider Toffoli gate [12], given by specification $(x, y, z) \rightarrow (x, y, z \oplus xy)$, where we fix input variable z to be 0, and consider first and third outputs to get the outputs of desired multiple output function output. The output y is considered to be "garbage" since it was not required in the function specification. The formal procedure to make a multiple output function reversible is described in [6].

We used the notation from the same paper. Here is a brief review.

Definition 1. Function $f(x_1, x_2, ..., x_n)$ of n Boolean variables is called **reversible** if: the number of outputs is equal to the number of inputs and any input pattern maps to a unique output pattern. ►

Definition 2. Garbage is the number of outputs added to make an n-input k-output Boolean function ((n, k) function) reversible. ►

In [6] we proposed a generalized Toffoli gate. The gate can take up to n inputs. $n-1$ outputs are equal to the corresponding inputs. The remaining line output is equal to the corresponding input or its inverse, depending on the other inputs. The general model is as follows:

- Take an (n, k) function and make it reversible by adding the theoretically minimal number of garbage outputs. The idea is to separate different output strings. If an output pattern appears m times, we need $\lceil \log m \rceil$ garbage outputs.

- Consider cascades of the following gates, where each horizontal line is of the following 4 types (their pictorial representation is shown in Fig. 1):

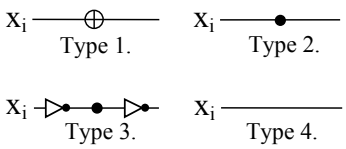

Figure 1: Horizontal line types.

1. Target line. Each gate has only one target line, appearing at some position j.
2. Positive control line. If the input is zero, the value of target line will not change. If it is one, the remaining positive/negative control lines will determine whether the value on the target line is negated.
3. Negative control line. If the input is one, the value of target line will not change. If it is zero, the other remaining positive/negative control lines will determine whether the value on the target line is negated.
4. Don't care line. The value of this line does not affect any output.

A vertical line intersects the horizontal lines of types 1-3. A typical gate is shown in the Fig. 2. In other words, for the given set of inputs $\{x_1, x_2, ..., x_n\}$, subset of variables $\{x_{i_1}, x_{i_2}, ..., x_{i_k}\}$, integer $j \in \{1, 2, ..., n\}, j \neq i_1, j \neq i_2, ..., j \neq i_k$ and set of $\leq k < n$ Boolean values $\{\sigma_1, \sigma_2, ..., \sigma_k\}$ the family consists of gates that leave all the bits unchanged, except for the j-th bit, whose value is $x_j \oplus x_{i_1}^{\sigma_1} x_{i_2}^{\sigma_2} ... x_{i_k}^{\sigma_k}$ Where x_i^{σ} is x_i if $\sigma = 1$

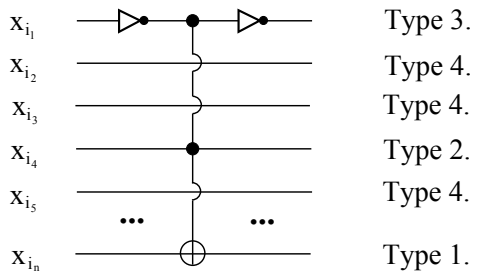

Figure 2: A typical gate

and $\bar{x_i}$ if $\sigma = 0$. If the term $x_{i_1}^{\sigma_1} x_{i_2}^{\sigma_2} ... x_{i_k}^{\sigma_k}$ consists of zero variables, we assign it a value of 1. Thus, each gate can be considered to be a generalized Toffoli gate—one bit is changing, according to the value of a product of inputs.

Due to the similarity of gates in the set Q and the generalized Toffoli gates, we introduce the following notation. The gate $Q_{a_1,a_2,...,a_n}$ is denoted as $\mathrm{TOF}s(x_{i_1}^{\sigma_1}, x_{i_2}^{\sigma_2}, ..., x_{i_{s-1}}^{\sigma_{s-1}}, y)$, which is constructed as follows:

- Write "TOF".
- Count the number of $a_i \in \{a_1, a_2, ..., a_n\}$ such that $a_i \neq 4$, write this number and opening bracket "(".
- For each a_i
  - if $a_i = 2$ write "x_i";
  - if $a_i = 3$ write "$\bar{x_i}$";
  - otherwise do nothing.

  Separate different entities with commas.
- Write x_j at the very end, where j is defined such that $a_j = 1$. Finish with the closing bracket ")".

For example, gate $Q_{3,1,4,2,4}$ can also be written as $\mathrm{TOF3}(\bar{x_1}, x_4, x_2)$.

Let Q be the set of all possible gates with n inputs. We have shown [6] that $|Q| = n3^{n-1}$. Given the model for function implementation, the problem of synthesis is to write a function in terms of a sequence of gates from the set Q. We solve this problem using an incremental approach. That is, we repeatedly chose a gate that will bring us closer to the desired function. In order to do this we need to be able to measure how close two functions are, we call this the distance between two functions. We then chose the gate that will decrease the distance between the realized function and the target function. We continue to do this until the distance is zero.

To give a formal definition of the distance, we need the following:

Definition 3. A **partial realization** of f is any function f' of the same set of variables. ▶

Definition 4. The **distance** between a reversible function f and its partial realization f' is the Hamming distance between their output parts of truth table. ▶

Definition 5. The **error** of the function f is its distance to the identity function. ▶

Example 1. The reversible function $f(x_1, x_2, x_3) = (x_1 \oplus \bar{x_2}x_3,\ \bar{x_2},\ x_1 \oplus x_2x_3)$ whose truth table is shown in Table 1 has 14 errors (shown in bold.)

x_1	x_2	x_3	f_1	f_2	f_3
0	0	0	0	**1**	0
0	0	1	**1**	**1**	**0**
0	1	0	0	**0**	0
0	1	1	0	**0**	1
1	0	0	1	**1**	**1**
1	0	1	**0**	**1**	1
1	1	0	1	**0**	**1**
1	1	1	1	**0**	**0**

Table 1: Truth table of $f(x_1, x_2, x_3) = (x_1 \oplus \bar{x_2}x_3,\ \bar{x_2},\ x_1 \oplus x_2x_3)$

Example 2. A partial realization $f'(x_1, x_2, x_3) = (x_1 \oplus \bar{x_2}x_3,\ x_2,\ x_3)$ of the reversible function from previous example is at distance 12 from the target function f (see Table 2.)

2.2 Synthesis Theory

The previous two examples had an even number of error bits and number of error 1-bits was equal to the number of error 0-bits. This result holds in general as shown in the following lemma.

f_1	f_2	f_3	f'_1	f'_2	f'_3
0	1	0	0	0	0
1	1	0	1	0	1
0	0	0	0	1	0
0	0	1	0	1	1
1	1	1	1	0	0
0	1	1	0	0	1
1	0	1	1	1	0
1	0	0	1	1	1

Table 2: Distance between f and its partial realization

Lemma 1. The error of a reversible function is even. Among the error bits the number of those equal to one is the same as the number equal to zero.

Proof. Write down the truth table of a reversible function and consider a column in the output part. Suppose the error in this column occurs in k 0 bits and s 1 bits. Since the function is reversible, each column of the output part of the truth table contains 2^{n-1} zero bits and 2^{n-1} one bits. Therefore, the chosen column has $2^{n-1}-k$ zeros and $2^{n-1}-s$ ones. Now, if we flip all the error bits of the function, it is the n-bit identity function (which also is reversible). From the point of view of number of zeros and ones this operation adds s zeros and k ones in the considered column. In the modified truth table numbers of zeros and ones become $2^{n-1}-k+s$ and $2^{n-1}-s+k$ correspondingly. Since the function is reversible, the number of zeros, as well as the number of ones should stay 2^{n-1}, which gives the following set of equations:

$$2^{n-1} - s + k = 2^{n-1} - k + s = 2^{n-1}$$

this is only possible for $k = s$. Therefore, the total number of errors in this column is $2k$, an even number. The same proof can be given for each of n output columns, thus the total error is an even number. ■

Note, that we actually proved a stronger statement, namely: the error is an even number in every output column. The other consequence one can derive, is that the distance between a function and its partial implementation is always an even number. It is also not hard to see that the number of ones that are out of place is the same as the number of zeros that are not at their place.

Consider the following simple idea for a synthesis method: start with identity function, find a gate from the set Q such that when added to the partial realization f' will decrease the distance to f. Sometimes there is no gate that decreases the distance to f. But at any time, it is possible to choose a gate that at least does not increase the distance to f. For an illustration see Example 3

Example 3. A reversible function with specification $(x, y) \to (\bar{y}, \bar{x})$ and the identity function, have this property: no gate will improve the distance function.

We use the word "step" to denote the addition of a gate to a cascade network. Therefore, the number of steps made is the number of gates in the network. A step is called positive (negative) if the distance increases (decreases).

Lemma 2. **(existence of non-positive step)** If the distance between f and f' is greater than zero (partial realization is not the function itself yet), there exists a gate in Q that transforms f' to f'' such that the distance between f and f'' is less than or equal to the distance between f and f'.

Proof. Let $(a_1, a_2, ..., a_{n-1}, X)$ be a string in the output part of the truth table of some partial realization with an error in bit X (which is assumed to be on the n-th place without loss of generality). Interchange it with the distance-1 string $(a_1, a_2, ..., a_{n-1}, a_n)$, where $a_n = \bar{X}$. To do so use the gate made of lines types $3 - a_1, 3 - a_2, ..., 3 - a_{n-1}, 1$ correspondingly, where the Boolean numbers $a_1, a_2, ..., a_{n-1}$ are treated as natural numbers. It is easy to see that the gate with such specification exchanges the named strings and does nothing to all others. Errors in the first (n-1) bits stay the same, but for the last bit the following table shows all the possible ways could happen when we make this interchange (Table 3). So, we get a zero step or a negative step. ■

Here, the question arises: is it possible to realize the function without positive steps? The answer is yes, and the design method is given in Theorem 3 of [6]:

Theorem 1. (upper bound) *Every reversible function can be realized with no more than* $n2^n$ *gates.*

The proof for this theorem is constructive and suggests a design procedure. The only thing that is left to prove, is that the steps "Increase order" and "Decrease order" are non-positive. Indeed, the essence of each of these steps is to put a correct bit (0 for "Increase order" and 1 for "Decrease order" steps) in its place. The Lemma above states

X value	a_n value	a_n was correct?	error was	error becomes	step is
0	1	Y	1	1	0
1	0	Y	1	1	0
0	1	N	2	0	-2
1	0	N	2	0	-2

Table 3: Effect of changing one bit

that each of these steps are non-positive only. This fact allows us to formulate the following result and use it as core to create a synthesis algorithm.

Theorem 2. *There exists a synthesis method that adds a gate only if it performs a non-positive change to the distance function. Such a method converges for any reversible input function.*

2.3 The Algorithm

Actual implementation of the algorithm works as follows:

1. Define the number $MaxMoves$ (which, in actual implementations was taken in the range of 50-500).

2. While the distance is greater than zero, from among all Q gates find best $MaxMoves$ steps. For each of them, find the best second step. After this step there are $MaxMoves$ pairs of gates in the list. Search for the sequence of 2 gates that maximally improve (minimizes) the distance between existing partial realization and the function itself. If such a pair is unique, attach first gate to the cascade and go back to 2.

3. If the two or more pairs of gates produce the same improvement to the distance, activate TieBraker: the function that finds the third best gate and if one of the pair has a better third gate (maximizes the decrease of the distance function), choose this pair. Then, attach first gate of the chosen pair to the cascade and go to 2.

4. If TieBraker was not able to find a pair where the third step gives better improvement, take the pair that gives the best improvement for the first gate. Go to 2.

5. If nothing of above worked, take the first pair of the gates among those that give the best improvement. Go to 2.

Theorem 2 states that the distance will not be increased, because there's always a zero step available. In general, such a method is not guaranteed to converge, although it does converge for every benchmark function we tried. We use this algorithm instead of the theoretical that is guaranteed to converge, since the latter is likely to give a larger number of steps. Since the distance can only decrease by at most two.

An (n, n) reversible function $(f_1, f_2, ..., f_n)$ in general can be realized by one of the $n!$ possible designs. This happens if we assume that the order of the output functions does not matter. We can enumerate the outputs in any order, therefore realize different functions. In our case, for the relatively large functions (starting from (9,9) functions and larger) we used a heuristics for the output permutation: we took the output permutation that gave the smallest error for the function or its complement. For the functions of smaller number of variables we are able to run all the possible permutations and choose the best result.

Example 4. Take function with specification $(x, y) \rightarrow (\bar{y}, \bar{x})$ from the previous example. Without the output permutation it will take us at least three gates to build a network for it: first step is a zero step, as it was shown in the previous example. Then, we have 2 errors in each of two output bits. This will require at least 2 more gates, since each of them in the best case scenario can take care of at most one output bit at a time. So, the theoretical minimum is 3 gates (in fact, our program does it in 3 steps). When a function $(x, y) \rightarrow (\bar{x}, \bar{y})$ with permuted outputs can be easily realized with two steps - subsequent negation of first and second input bits.

2.4 Multiple Output Functions

Using the results of [9] we are able to add the minimal number of garbage bits in order to make a multiple output function reversible. The benefit in realization of a multiple output function, is that we do not care about some of the outputs: the

values of the garbage is of no interest. This allows to:

1. Have a smaller error - therefore, in general, have less steps to make in order to create a network.

2. Have more freedom in changing "don't care" outputs. There is no risk in adding an error to a "don't care" output. We minimize the distance to the "care" outputs only.

name	in/out	Miller	We
ex1	3	3	3
ex2	3	5	5
ex3	4	7	7
ex4	3	4	3
ex5	4	5	4
ex6	4	12(10)	7
ex7	4	9(7)	7

Table 4: Comparison with Miller's results

3 Benchmarks

In this section we compare our algorithm to the previous algorithms. Unfortunately, some authors do not give enough information to allow us to do so. For example, authors of [4] suggest an approach for reversible cascade synthesis of one output functions. They do not provide the experimental results, nor the algorithm, therefore we can not compare those results to ours. Shende *et al.* [11] provide the optimal synthesis method for the $(3, 3)$ reversible functions only. Work [10] is concentrated on reversible synthesis of symmetric functions, which is less general than our approach. Iwama *et al.* [3] base their method on circuit transforms, but they to not provide any experimental data.

We compare our results with the results of the two systematic methods one by Miller [7] and the other by Mishchenko and Perkowski [9].

Miller [7] suggests a reversible function synthesis that starts with a reversible specification only. He uses a spectral technique to find the best gate to be added in terms of gates (NOT, CNOT, Toffoli3, and Toffoli4) and adds the gate in a cascade-like manner. This method has been modified in [8] to handle generalized Toffoli gates. In his method the output function is required to appear as a set of actual outputs or their negations. Miller also used a post processing process to simplify the network (the results of simplification are given in brackets for the cases the process was done). The results of comparison of all examples from [7] to ours are summarized in Table 3, where **name** is a name of the benchmark function, **in/out** - number of its inputs/outputs, **Miller** - number of gates for Miller's method, **We** - number of gates for the proposed method.

Example 5. Since our method gives a better result for *ex4.pla*, we'd like to show our network for this function. *ex4.pla* is a $(4, 4)$ reversible function, given as the truth vector $[3, 11, 2, 10, 0, 7, 1, 6, 15, 8, 14, 9, 13, 5, 12, 4]$ whose binary representation gives the actual Boolean values. Using the output permutation $(2, 1, 3, 4)$, the scheme consists of the following seven gates: TOF1(b) TOF3($\bar{c}, d, a$) TOF3(a, d, c) TOF3($a, \bar{c}, d$) TOF3($\bar{c}, \bar{d}, a$) TOF3($\bar{a}, \bar{d}, c$) TOF3($\bar{a}, \bar{c}, b$) for the names of variables a, b, c and d. In comparison, Miller's synthesized network is: TOF1(d) TOF1(b) TOF3(b, c, d) TOF1(b) TOF3(a, c, d) TOF1(c) TOF3(a, b, d) TOF2(a, c) TOF3(b, c, a) TOF2(b, c) TOF2(a, c) TOF(a, b). This networks was transformed to the following: TOF1(d) TOF1(b) TOF3(b, c, d) TOF1(b) TOF3(a, c, d) TOF1(c) TOF3(a, b, d) FRE(b, c, a) TOF2(b, c) TOF(a, b), where FRE(b, c, a) is the Fredkin gate [2] built on variables b, c and a correspondingly.

Mishchenko and Perkowski [9] suggest a reversible wave cascade method and evaluate the complexity of some benchmark functions in terms of the number of these cascades. They don't provide the actual design for the described method, instead they give upper bounds, we compare their results to ours and summarized the comparison in Table 3. Although our results in terms of the total complexity aren't always better than those of Mishchenko and Perkowski, the important factor - number of garbage bits is definitely beneficial in our approach. We were not able to compare the results for functions of larger number of inputs/outputs due to the huge amount of work our program needs to find a network representing such a function. In this table first three columns are name, number of input and output bits of a benchmark function. Columns **NRC** and **RCG** list the number of cascades and garbage (calculation of garbage is taken from [6]) of Mishchenko and Perkowski's method; the remaining two columns are number of gates and number of garbage outputs in our method and proposed design corre-

name	in	out	NRC	RCG	C	G
5xp1	7	10	31	38	49	0
9sym	9	1	51(52?)	60	56	9
rd53	5	3	14	19	13	4
rd73	7	3	36	43	36	6

Table 5: Comparison with Perkowski's results

spondingly. Our method does not alway find the realization with the minimum number of gates, but if we consider the complexity of a benchmark function to be sum of number of gates and the garbage, then our method gives a better result. For example, "rd53.pla" can be realized in terms of an ESOP with 14 terms as the result of Perkowski and Mishchenko states. For our method this number is 13, which shows that the proposed method can do better than EXOR minimization. The following example contains one more function for which our method is more beneficial, compared to the standard non-reversible EXOR PLA.

Example 6. The $(5,1)$-function $2of5.pla$ whose output is one iff exactly two of the input variables are one in terms of EXOR PLAs can be realized with 8 terms, when our synthesis method is capable of creating a network (for the proposed structure) with 7 gates only. The function $2of5.pla$ is not balanced, therefore the minimal garbage for it is 5. Thus, the $(5,1)$-function becomes a $(6,6)$ reversible function. We used the last output to realize the function, and named the inputs as a, b, c, d, e, and f, where the last input is a constant 0. The network structure is as follows: TOF5$(a, \bar{d}, e, \bar{f}, c)$ TOF4$(\bar{b}, c, \bar{d}, f)$ TOF4$(\bar{a}, c, \bar{e}, f)$ TOF5$(\bar{a}, b, d, \bar{e}, f)$ TOF3$(\bar{a}, \bar{f}, e)$ TOF5$(\bar{b}, \bar{c}, d, \bar{e}, f)$ TOF5$(b, \bar{c}, \bar{d}, \bar{e}, f)$.

4 Future Work

The algorithm has its shortcomings. There are not enough theoretical results to build a good synthesis algorithm, and there is a lack of heuristics. A better understanding of the theory is required, coupled with heuristics to reduce the search space. Here are some approaches that warrant further investigation:

- Apply spectral techniques to measure the complexity of a function. This may be a better metric than than the distance we use here. However, the results in this direction [8] are not better than ours.
- Good moves may be detected from a decision diagram representation of the function.
- The problem of the output permutation has not been solved. In Miller's method [8] this is determined automatically—each input converges to one output (or its complement). Can spectral techniques help us here?
- For each step we investigate $n3^{n-1}$ possible gates. This is only feasible for small values of n. We need some heuristics to reduce this number.

5 Conclusion

The synthesis method presented in this paper can realize multiple output functions with minimal garbage outputs. The results are similar to other methods which must start with a reversible specification [8] or require a much larger number of garbage outputs [9]. It can also handle incompletely specified functions. If the function is not specified for a given input assignment, then the output does not affect the distance functions. This is surprising, because incompletely specified functions are not easily handled with EXOR minimizations and the Toffoli gate is closely related to the EXOR gate.

References

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

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

[3] K. Iwama, Y. Kambayashi, and S. Yamashita. Transformation rules for designing cnot-based quantum circuits. In *Proceedings of the Design Automation Conference*, New Orleans, Louisiana, USA, June 10-14 2002.

[4] A. Khlopotine, M. Perkowski, and P. Kerntopf. Reversible logic synthesis by iterative compositions. *International Workshop on Logic Sysnthesis*, 2002.

[5] R. Landauer. Irreversibility and heat generation in the computing process. *IBM J. Res.*, 5:183–191, 1961.

[6] D. Maslov and G. W. Dueck. Garbage in reversible design of multiple output functions. In *6th International Symposium on Representations and Methodology of Future Computing Technologies*, March 2003.

[7] D. M. Miller. Spectral and two-place decomposition techniques in reversible logic. In *Midwest Symposium on Circuits and Systems*, Aug. 2002.

[8] D. M. Miller and G. W. Dueck. Spectral techniques for reversible logic synthesis. In *6th International Symposium on Representations and Methodology of Future Computing Technologies*, March 2003.

[9] A. Mishchenko and M. Perkowski. Logic synthesis of reversible wave cascades. In *International Workshop on Logic Sysnthesis*, June 2002.

[10] M. Perkowski, P. Kerntopf, A. Buller, M. Chrzanowska-Jeske, A. Mishchenko, X. Song, A. Al-Rabadi, L. Joswiak, A. Coppola, and B. Massey. Regularity and symmetry as a base for efficient realization of reversible logic circuits. In *International Workshop on Logic Sysnthesis*, 2001.

[11] V. V. Shende, A. K. Prasad, I. L. Markov, and J. P. Hayes. Reversible logic circuit synthesis. In *ICCAD*, San Jose, California, USA, Nov 10-14 2002.

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