

# Reversible Cascades with Minimal Garbage

Dmitri Maslov and Gerhard W. Dueck, *Member, IEEE*

**Abstract**—The problem of minimizing the number of garbage outputs is an important issue in reversible logic design. We start with the analysis of the number of garbage outputs that must be added to a multiple output function to make it reversible. We give a precise formula for the theoretical minimum of the required number of garbage outputs. For some benchmark functions, we calculate the garbage required by some proposed reversible design methods and compare it to the theoretical minimum. Based on the information about minimal garbage, we suggest a new reversible design method that uses the minimum number of garbage outputs. We show that any Boolean function can be realized as a reversible network in terms of this new approach by giving the theoretical method of finding such a network. Using a heuristics synthesis approach, we create a program and run it to compare results of our synthesis to the previously reported synthesis results for the benchmark functions with up to ten variables. Finally, we show that the synthesis for the proposed model can be accomplished with lower cost than the synthesis of EXOR PLAs.

**Index Terms**—Logic synthesis, reversible logic, minimal garbage, quantum computing.

## I. 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 [9] 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.

Reversible logic has applications in nanotechnology, quantum computing, low power CMOS, optical computing, and DNA computing. One of the important applications of the reversible logic is quantum computation. Quantum computations are known to solve some exponentially hard problems in polynomial time [15]. All quantum computations are necessarily reversible. Therefore research of reversible logic is beneficial to the development of future quantum technologies: reversible design methods might give rise to methods of quantum circuit

construction, resulting in much more powerful computers and computations.

Most gates used in digital design are not reversible. For example the AND, OR, and EXOR 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 [3], Toffoli [20], and Fredkin [4] gates. These gates have been studied in detail. However, good synthesis methods have not emerged. Shende *et al.* [17], [18] presented an exhaustive method that produces a minimal circuit with up to 3 input variables. In addition they presented a synthesis method for larger functions that relies on gate libraries. Iwama *et al.* [5] describe transformation rules for CNOT based circuits. These rules may be of use in a synthesis method. Miller [11] uses spectral techniques to find near optimal circuits. Results were reported for 3 and 4 variable reversible functions. Mishchenko and Perkowski [14] 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 [14] has not been implemented. A regular symmetric structure has been proposed by Perkowski *et al.* [16] to realize symmetric functions. Khan and Perkowski [6], [7] introduce a new family of gates and describe a synthesis method for this new family. The aim of their method is to minimize the gate count.

Traditional design methods use, among other criteria, the number of gates as complexity measure (sometimes taken with some specific weights reflecting the area of the gate). From the point of view of reversible logic we have one more factor, which is 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. Some methods [6], [7], [14] propagate the input information to the outputs, therefore introducing constant inputs and garbage outputs—information that is not needed for the computation. In some cases garbage is unavoidable. For example, a single output function of  $n$  variables will require at least  $(n - 1)$  garbage outputs, since the reversibility necessitates an equal number of outputs and inputs.

The importance of minimizing the number of garbage outputs is illustrated with the following example. Say we want to realize a 5 input 3 output function as a reversible circuit for a quantum computer, but the design requires 7 additional garbage outputs, resulting in a 10-input 10-output reversible function. In the year 2003 the best quantum computer we

Manuscript received ???, 2003; revised ???, 2003; revised ???, 2003. The work of G. W. Dueck was supported in part by a grant from the Natural Sciences and Engineering Research Council of Canada.

D. Maslov is with the Department of Computer Science, University of Victoria, Victoria, BC, Canada, V8W 3P6 (dmitri.maslov@unb.ca).

G. W. Dueck is with the Faculty of Computer Science, University of New Brunswick, Fredericton, NB, Canada, E3B 5A3 (gdueck@unb.ca).

have is a 7 qubit computer [1], therefore it will not be possible to implement such a design. In other words, in case of choosing between increase of the number of garbage outputs and increase of the number of gates to be used in a reversible implementation, the preference should be given to the design method delivering the minimum garbage. In this case we will be able to build the device, while it is impossible with the other method.

We propose a structure and a systematic design method that require the minimal number of garbage outputs. This work focuses on analyzing the conditions for minimal garbage, introduces and analyzes the model. The synthesis of reversible functions differs from the conventional logic synthesis, the following restrictions apply: fan-out and feed back are not permitted [15].

The results of this paper may be applied in any of the technologies that support reversible computations. However, in a few places we mention quantum technologies as information on it is easily accessible, which makes it possible to do some calculations that help to estimate the real cost of an implementation. Our algorithm would only be applicable to quantum minimization if the set of building blocks is restricted to Toffoli gates. Additional concerns might be of interest for quantum computing, in particular minimizing the critical path gate count. In general, the paper is not tied to any specific technology.

The remaining part of the paper is organized as follows. In Section 2 we give basic definitions of reversible logic theory. In Section 3 we analyze conditions for minimal garbage and inspect the number of garbage bits in existing reversible synthesis methods of multiple output functions. We introduce a new reversible synthesis model in Section 4 and show some theoretical results, such as a regular synthesis procedure. The theoretical synthesis method is likely to produce large circuits. Therefore, a heuristic synthesis method is developed and tested on some benchmark functions in Section 5. We conclude the paper by showing that the optimal reversible synthesis in our model is beneficial in comparison to the synthesis of EXOR polynomials.

## II. BASIC DEFINITIONS

*Definition 1:* The  $n$ -input  $k$ -output Boolean function  $f(x_1, x_2, \dots, x_n)$  (referred to as  $(n, k)$  function) is called reversible if:

- the number of outputs is equal to the number of inputs;
- each input pattern maps to a unique output pattern.

In other words, reversible functions are those that perform permutations of the set of input vectors. We illustrate the need for garbage outputs and/or constant inputs with the following example.

*Example 1:* Consider function  $(x, y) \rightarrow xy$  (where concatenation denotes the logical AND operation). It is impossible to make it reversible by adding a single output. One of the ways to make it reversible is adding one input and two outputs so that the function becomes as shown in Table I. The output vector of the desired function can be observed in the third output column of the table when the value of variable  $z = 0$

TABLE I  
REVERSIBLE FUNCTION COMPUTING THE LOGICAL AND

$x$	$y$	$z$	$x$	$y$	$z \oplus xy$
<b>0</b>	<b>0</b>	0	0	0	<b>0</b>
0	0	1	0	0	1
<b>0</b>	<b>1</b>	0	0	1	<b>0</b>
0	1	1	0	1	1
<b>1</b>	<b>0</b>	0	1	0	<b>0</b>
1	0	1	1	0	1
<b>1</b>	<b>1</b>	0	1	1	<b>1</b>
1	1	1	1	1	0

(shown in bold font). To realize the function, the input  $z$  must be the constant zero, and two garbage outputs are present. The Toffoli gate [20] realizes this function.

The previous example shows the necessity of adding inputs and/or outputs to make a function reversible. This leads to the following definition.

*Definition 2:* **Garbage** is the number of outputs added to make an  $(n, k)$  function function reversible.

Our definition of garbage differs from [4], [6], [7], [14] where the set garbage outputs does not include the inputs of the function if they are passed unchanged. However, for the analysis in this paper our definition is more practical. Let us illustrate this with the following example. Given information on the number garbage outputs and the number of outputs of the function, with our definition the total number of variables can be easily found. In terms of the other garbage definition the values of the outputs and garbage bits have to be computed for all possible inputs in order to calculate the total number of variables in the circuit.

We use the term “constant inputs” to denote the inputs that are added to an  $(n, k)$  function to make it reversible. In the previous example a single constant input was added, namely the variable  $z$ . The meaning of the prefix “constant” of the term is easy to see from the same example. The target output is realized when the constant input is 0.

The following simple formula shows the relation between garbage outputs and constant inputs

$$input + constant\ inputs = output + garbage\ bits.$$

There are many ways of making a multiple output Boolean function reversible, each requiring a different number of garbage outputs to be created. We start by analyzing the conditions for the number of garbage outputs. We analyze the number of garbage bits for existing synthesis methods. A conclusion of this analysis can be summarized in a few words: the number of garbage is excessive, therefore a different approach/model should be created. There are some restrictions on the new model. First, and very important, is few garbage outputs (in fact, it will be the theoretically minimum number). Second, the model gates should have a reasonable cost if implemented in at least one of the technologies that support reversible logic implementations. Finally, the results of the actual synthesis should not be large in comparison to the other synthesis method results. If such model is created, it may be very important for evolving further reversible logic theory and bringing its theoretical results to an actual technology.

### III. MINIMAL GARBAGE

Before we analyze the number of garbage outputs in other models, we need to show a formula to calculate the minimum number of garbage bits required for any reversible synthesis procedure independently of its nature.

*Theorem 1:* For an  $(n, k)$  function the minimum number of garbage bits required to make it reversible is  $\lceil \log(M) \rceil$ , where  $M$  is the maximum of number of times an output pattern is repeated in the truth table.

*Proof:* The output of a reversible function is a permutation of its input. Therefore, the obstacle in having a multiple output function being reversible is that some output pattern appears more than once. In order to separate these outputs we have to introduce new inputs to assign additional bits to the output vector. If an output  $(o_1, o_2, \dots, o_k)$  has the largest occurrence in the output vector and it appears  $M$  times, then in order to separate different occurrences of it we need to introduce  $\lceil \log(M) \rceil$  new output bits.  $\lceil \log(M) \rceil$  new bits will be capable of creating  $2^{\lceil \log(M) \rceil} \geq M$  new patterns. And, since the output  $(o_1, o_2, \dots, o_k)$  had the largest occurrence among all other outputs, all other outputs can be easily separated from one another by means of  $\lceil \log(M) \rceil$  bits. ■

#### A. Analysis of Garbage in Existing Methods

In this subsection we analyze number garbage bits in proposed reversible designs for non-reversible multiple-output functions. Several of the proposed design methods (for example [8], [17], [18], and [11]) start with a reversible function, we will not deal with them. The garbage bits are introduced in a preprocessing phase, during which the function is made reversible. Note that there are many ways in which the value of the garbage outputs can be set. Different settings of these variables will lead to results with varying complexity. Below, we concentrate on the analysis of the garbage in reversible synthesis methods for multiple output functions.

Mishchenko and Perkowski [14] suggest a reversible wave cascade. The design is shown in Fig. 1A. For the purposes of garbage analysis here we concentrate only on the number of garbage outputs added. Trivial analysis of the number of garbage bits shows that in the proposed model the garbage size will be  $(n + M)$ , where  $n$  is the number of inputs of the multiple output function  $f$  and  $M$  is the number of cascades (Maitra terms) in the particular realization of a function.

Perkowski *et al.* [16] suggest a regular structure for a symmetric  $(n, k)$  function reversible design, called RPGA (Fig. 1B). The synthesis for a symmetric function, as it is easy to see from the structure Fig. 1B, will require number of garbage bits that is equal to the sum of the number of inputs and the number of gates used (additional wires are reserved for the outputs), which gives

$$n + \frac{n(n-1)}{2} = \frac{n(n+1)}{2}.$$

Khan and Perkowski [6], [7] propose a method that has a similar structure to the one described in [14]. The synthesis and garbage results for these methods are essentially the same, although later work has, on average, worse results both in the number of gates and the number of garbage bits.

TABLE II  
NUMBER OF GARBAGE OUTPUTS IN DIFFERENT MODELS

name	in	out	RWCG	RPGAG	KPG	max OO	MG
5xp1	7	10	38	> 28	53	1	0
9sym	9	1	60	45	60	420	9
b12	15	9	43	> 120	41	6944	13
clip	9	5	72	> 45	N/A	37	6
in7	26	10	61	> 351	N/A	11651840	24
rd53	5	3	19	15	19	10	4
rd73	7	3	43	28	47	35	6
rd84	8	4	66	36	68	70	7
sao2	10	4	38	> 55	52	513	10
t481	16	1	29	> 136	28	42016	16
vg2	25	8	209	> 325	217	12713984	24

We calculated the number of garbage bits in the proposed models for some benchmark functions. Table II summarizes the result for the methods suggested in [7], [14], [16] for some benchmark functions used in [14]. The first column shows the name of the function, the second and third are the number of input and output bits respectively. The fourth column (**RWCG**) is the number of garbage bits in the wave cascade model. Column **RPGAG** shows the number of garbage bits for the RPGA method given by the formula described above. Since every non-symmetric function can be made symmetric by adding new outputs, the procedure of making the function reversible can be done prior to the usage of the algorithm as suggested by Perkowski *et al.* [16]. In general, such a procedure requires many additional inputs, each resulting in a high garbage price for their introduction. In cases where the function is not symmetric we use the sign “>” to represent that the actual garbage output count is higher. Numbers in the sixth column (**KPG**) represent the garbage cost for the Khan family gates synthesis. The seventh column shows the maximal output occurrence, the logarithm of which added to the function input size forms the last column—the minimal number of garbage bits to be added to make the corresponding function reversible.

We conclude this subsection with the observation that all three regular methods analyzed have garbage that is far from the theoretical minimum. In the next section we introduce a new regular structure with better garbage characteristics; in fact, the number of garbage outputs is the theoretical minimum.

### IV. A NEW STRUCTURE: REVERSIBLE CASCADES WITH MINIMAL GARBAGE

#### A. Definition of the Model

We consider the set of model gates which is based on the generalized Toffoli gate with additional negated controls. We use the same pictorial representation, and take  $(n, n)$ -gates where each horizontal line is one of the following 4 types (Fig. 2):

- 1) Target line. Each gate has a single target line appearing at some position  $j$ .
- 2) Positive control line. If the input on this line is zero, the value of the target line will not change. If the input is one, the other positive/negative control lines determine whether the value on the target line is negated.

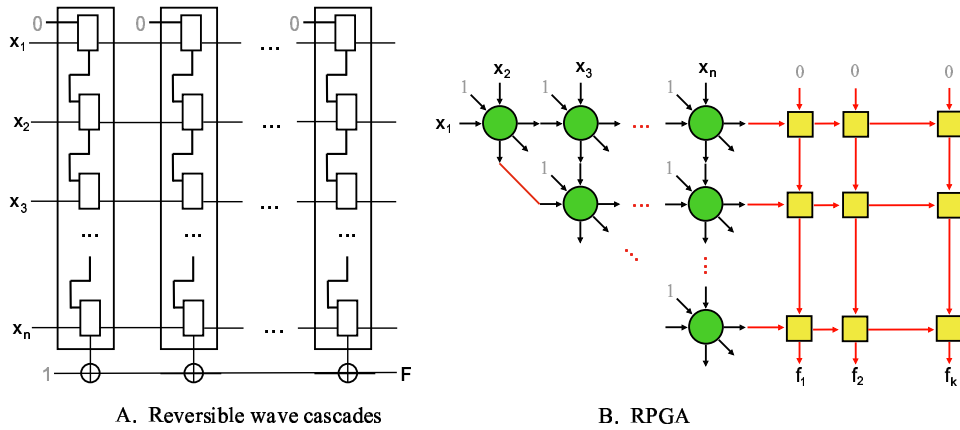


Fig. 1. Two reversible design structures.

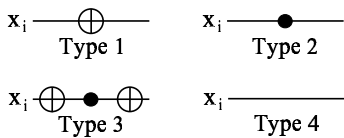


Fig. 2. Horizontal line types.

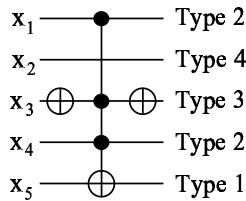


Fig. 3. A single gate.

TABLE III  
TRUTH TABLE FOR EXAMPLE 2

$x_1$	$x_2$	$x_3$	$f_1$	$f_2$	$f_3$
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

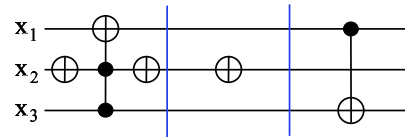


Fig. 4. Circuit used in Example 2.

- 3) Negative control line. If the input on this line is one, the value of target line will not change. If it is zero, the remaining positive/negative control lines determine whether the value on the target line is negated.
- 4) Don't care line. The value on this line does not affect any output.

The vertical line intersects horizontal lines of types 1-3. In other words, for the given set of inputs  $\{x_1, x_2, \dots, x_n\}$ , the subset of variables  $\{x_{i_1}, x_{i_2}, \dots, x_{i_k}\}$ , integer  $j \in \{1, 2, \dots, n\}$ ,  $j \neq i_1, j \neq i_2, \dots, j \neq i_k$  and set of  $1 \leq k < n$  Boolean numbers  $\{\sigma_1, \sigma_2, \dots, \sigma_k\}$  the family consists of gates that leave all the bits unchanged, except for the  $j$ -th bit, whose value is  $x_j \oplus x_{i_1}^{\sigma_1} x_{i_2}^{\sigma_2} \dots x_{i_k}^{\sigma_k}$ . If the term  $x_{i_1}^{\sigma_1} x_{i_2}^{\sigma_2} \dots x_{i_k}^{\sigma_k}$  consists of zero variables, we assign it a value of 1.

The graphical representation of a gate is shown in Fig. 3.

The network we want to build is a cascade consisting of the set of described gates.

*Example 2:* Take a reversible function  $(x_1, x_2, x_3) \rightarrow (x_1 \oplus \bar{x}_2 x_3, \bar{x}_2, x_1 \oplus x_2 x_3)$  (output is written as a set of minimal length EXOR polynomials). The fact that the function is reversible is easy to see from its truth table shown in Table III. A possible implementation is shown in (Fig. 4).

Further, we refer to this model as RCMG (Reversible Cascades with Minimal Garbage).

*B. Quantum Cost Analysis*

Quantum technology (technologies) is one of several technologies that uses reversible gates and computations. So, it is interesting to analyze the generic quantum cost of the introduced model to see whether and how it differs from the costs of the gates used by other models.

Quantum transformations are necessarily reversible, this follows from the only condition used to determine whether a transformation can be accomplished: it must be a unitary operator on the set of amplitudes [15]. This condition does not imply how difficult it is to realize a given unitary transformation; it only states the theoretical possibility. In this paper, we refer to the quantum cost as the number of 1-qubit or two qubit controlled-V operations needed to construct the proper gate (see [2], [15] for more information). This is only a generic definition that gives us a crude approximation of the actual cost.

In conjunction with reversible logic synthesis, the following transformations can be realized as one gate with unit quantum cost:

- NOT gate (also known as quantum X gate). For Boolean values it acts as the conventional NOT gate.
- CNOT gate, which acts as  $TOF(x_1; x_2)$ . In other words, in the Boolean case it flips  $x_2$  iff  $x_1 = 1$ .

The set of gates NOT, CNOT is not complete since they only realize linear functions. Thus, in order to make the set complete (as a set of Boolean operators), the Toffoli gate [20],  $TOF(x_1, x_2; x_3)$  was added. Unfortunately, this gate cannot be realized as a single 1-bit or controlled-V operation. A quantum realization with cost 5 was found, and it is likely that this is the minimum. The more controls a generalized Toffoli gate has, the higher its cost in terms of the number of elementary quantum transformations required.

The problem of building quantum blocks to realize the Toffoli gates was investigated by many authors. For a comparison of quantum costs of the Toffoli and RCMG model gates we will use the results from [2]. For other implementations the costs can easily be recalculated.

*Definition 3:* The quantum cost of a gate  $G$ ,  $|G|$  is the number of basic operations (1-bit and controlled-V type [2]) required to realize the function given by  $G$ .

No particular realization of a gate (for most of the gates) was proven to be optimal, so the numeric value of the quantum cost may change as soon as better gate realizations are proposed.

To analyze the quantum cost of an RCMG model gate, we suggest starting with its simplification, and then comparing its cost to the cost of a generalized Toffoli gate. Note, that an RCMG model gate can be considered as a generalized Toffoli gate and a set of NOT operations. First, we should try to minimize the number of NOTs in the circuit.

The following method of eliminating NOT gates from the structure can be used. In some designs, like the one shown in Fig. 3, two NOT gates may be adjacent. Therefore, they are redundant. For the example shown in Fig. 4 pruning such NOT gates gives the design shown in Fig. 5. In general, we

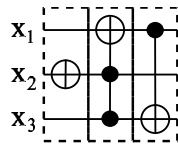


Fig. 5. The pruned circuit was obtained by eliminating adjacent NOT gates from the circuit shown in Fig. 4.

divide any gate from the set  $Q$  into three logical parts:

- First NOT array: the set of all NOT gates before the vertical line.
- AND-EXOR array (the generalized Toffoli gate): the set of all AND and EXOR gates on the vertical line.
- Last NOT array: the set of all remaining NOT gates.

The general rule for pruning NOT gates is as follows:

- 1) Define TEMP array as an array of NOT gates of length  $n$ , such that there is at most one NOT gate in each position. Initially no NOT gate is present in the TEMP array.
- 2) Starting from the beginning of a particular network, keep NOTs from the first NOT array of a first  $Q^1 \in Q$  gate together with the next AND-EXOR array and call this

TABLE IV  
GATE COST COMPARISON

# of controls	Garbage	Toffoli gate cost	RCMG gate cost $\leq$	Rel. cost $\leq$	Average rel. cost
2	0	5	7	1.4	1.2
3	0	13	16	1.231	1.115
4	0	29	33	1.138	1.069
5	0	61	66	1.082	1.041
6	0	125	131	1.048	1.024
6	4	112	118	1.054	1.027
7	0	253	260	1.028	1.014
7	3	124	131	1.056	1.028
8	0	509	517	1.016	1.008
8	4	172	180	1.047	1.023

structure a block. The last NOT array of gate  $Q^1$  is called TEMP. If  $Q^1$  was one of  $\bar{x}_1, \bar{x}_2, \dots, \bar{x}_n$ , add it to TEMP.

- 3) Take next gate  $Q^2 \in Q$  from the network. If  $Q^2 \notin \{\bar{x}_1, \bar{x}_2, \dots, \bar{x}_n\}$  update the TEMP array by computing its exclusive or with the first NOT array of  $Q^2$  that is, keep the modulo-2 sum of number of NOTs at each wire. If the NOT gate from the TEMP array meets a target line or a “don’t care” line, it can be passed through the gate, so delete these occurrences from the TEMP array and add them to the output array of the  $Q^2$  gate. Unite the TEMP array with the AND-EXOR array (create a new block), let  $Q^1 := Q^2$  and go to step 2. If the  $Q^2$  gate was one of gates  $\bar{x}_1, \bar{x}_2, \dots, \bar{x}_n$ , update TEMP array by computing the “exclusive or” of TEMP with  $Q^2$ , let  $Q^1 := Q^2$  and go to step 2.
- 4) When the last gate in the network is reached, create the last block by attaching the TEMP array.

It is easy to see that the network consisting of the described blocks is equivalent to the network built from the gates  $Q$ . The number of blocks of the pruned network is the number of gates of the initial  $Q$ -network minus the number of gates from the set  $\{\bar{x}_1, \bar{x}_2, \dots, \bar{x}_n\}$  of this  $Q$ -network plus the TEMP array. Therefore, both the set of NOT gates and the length of the structure can only be decreased.

The new gate will consist of the NOT array in front of the Toffoli gate, where the NOTs may appear only in front of the control lines, which makes it easy to compare the costs. The result of this comparison is summarized in Table IV. It is important to notice that the cost of the gate in the new model differs from the cost of the widely used generalized Toffoli gate only marginally. For example, the quantum cost of a Khan gate [6], [7] with  $k$  controls is equivalent to the cost of the two Toffoli gates, which, in a rough calculation, should multiply the cost of a circuit by 2 when compared to the synthesis results of RCMG model.

**Note** that the NOT pruning procedure is post processing, that does not affect the synthesis.

### C. Theoretical Synthesis

In order to formulate and prove some results we need to enumerate the set of all gates considered in the structure. Every gate can be uniquely specified by describing the set of horizontal lines. From now on, we will use the notation

$Q_{a_1, a_2, \dots, a_n}$  for the gate consisting of wire types  $a_1, a_2, \dots, a_n$  in order of appearance from top to bottom.

**Lemma 1:** The set of all possible gates in the proposed structure consists of  $n * 3^{n-1}$  elements.

**Proof:** Distribute lines among the  $n$  places we have to fill in order to define a gate. Initially, there are  $n$  places for the target line; after assigning it, there are  $(n - 1)$  places left to be occupied by positive, negative and “don’t care” lines to be placed in any combination. The number of ways to put them, therefore, is  $3^{n-1}$ . This gives a total of  $n * 3^{n-1}$  different gates. ■

**Theorem 2: (lower bound)** There exists a reversible function that requires at least  $\frac{2^n}{\ln 3} + o(2^n)$  gates.

**Proof:** The number of all reversible functions of  $n$  variables is  $2^{2^n}$  (as the number of permutations of  $2^n$  elements). The number of different transformations produced by gates of the set  $Q$  is  $n3^{n-1}$ . Assuming that taking some of the gates and building networks with different order produces different reversible functions (which is not always true, since, for instance, two consecutive gates  $Q_{1,4,4,\dots,4}$ , or  $\bar{x}_1$  do nothing), we get a complexity for the hardest function of  $\log_{n3^{n-1}}(2^{2^n})$ . This means that there exists a reversible function which can be realized with a complexity not less than  $\log_{n3^{n-1}}(2^{2^n})$ . Using the formula  $\ln(k!) = k \ln k - k + o(k)$  for  $k = 2^n$  write:

$$\begin{aligned} \log_{n3^{n-1}}(2^{2^n}) &= \frac{\ln(2^{2^n})}{\ln(n3^{n-1})} = \frac{2^n \ln 2^n - 2^n + o(2^n)}{\ln(3^{n-1}) + \ln(n)} \\ &= \frac{n2^n - 2^n + o(2^n)}{(n-1)\ln 3 + \ln(n)} = \frac{(n-1)2^n + o(2^n)}{(n-1)\ln 3 + \ln(n)} \\ &= \frac{2^n + o(2^n/n)}{\ln 3 + \frac{\ln(n)}{n-1}} = \frac{2^n}{\ln 3} + o(2^n). \end{aligned}$$

**Theorem 3: (upper bound)** Every reversible function can be realized with no more than  $n2^n$  gates.

**Proof:** We use an idea similar to bubble sorting in our constructive proof.

First, note that the set of gates that do not have a “don’t care” line, i.e. the set  $Q' = \{Q_{a_1, a_2, \dots, a_n} | a_1, a_2, \dots, a_n \in \{1, 2, 3\}, \text{ and there exists a unique } a_i = 1\}$  interchange the two output strings  $(3 - a_1, 3 - a_2, \dots, 3 - a_{i-1}, x, 3 - a_{i+1}, \dots, 3 - a_n)$  and  $(3 - a_1, 3 - a_2, \dots, 3 - a_{i-1}, \bar{x}, 3 - a_{i+1}, \dots, 3 - a_n)$  in the output part of the truth table (natural numbers 0 and 1 should be treated as Boolean 0 and 1 respectively). This also means that a single gate changes the two Hamming distance-one strings in the output part of the truth table.

Second, we define a special total order on the set  $Q'$  of gates. In this order:

- strings with a fewer number of ones precede (denoted as  $\prec$ ) those with a larger number of ones;
- strings with an equal number of ones are arranged in lexicographical order.

In other words, the order is as follows:  $(0, \dots, 0, 0) \prec (0, \dots, 0, 1) \prec (0, \dots, 0, 1, 0) \prec \dots \prec (1, 0, \dots, 0, 0) \prec (0, \dots, 0, 1, 1) \prec (0, \dots, 0, 1, 0, 1) \prec \dots \prec (1, 0, \dots, 0, 1) \prec \dots \prec$

$(1, 1, \dots, 1, 0) \prec (1, \dots, 1, 1)$ . We will also use standard order on Boolean constants:  $0 \prec 1$ .

The method is to copy the input part of the truth table to the output part, which corresponds to the situation when no network is built yet, therefore the output is equal to the input. Then, apply operations defined by the gates from the set  $Q'$  to bring each string to its place, starting from the string with the lowest order and finishing with the string with the highest order.

Take any string  $(a_1, a_2, \dots, a_n)$  and bring it to its place. If the string is already at its place, we are done. If it is not, then since we are moving the strings in ascending order, its place is occupied by a string of higher order. This is true, since by induction the strings of lower order are already at their places and no string is repeated. Therefore, the place of  $(a_1, a_2, \dots, a_n)$  is occupied by a  $(b_1, b_2, \dots, b_n)$ . Compose string  $(a_1 \vee b_1, a_2 \vee b_2, \dots, a_n \vee b_n)$ .

**Step 1: increase the order of the target.** Take the string  $(b_1, b_2, \dots, b_n)$ , find minimal  $i$ , such that  $a_i = 1$  and  $b_i = 0$  and exchange distance-one strings  $(b_1, b_2, \dots, b_n)$  and  $(a_1 \vee b_1, a_2 \vee b_2, \dots, a_i \vee b_i, b_{i+1}, \dots, b_n)$ . Now, the place where we wanted to see  $(a_1, a_2, \dots, a_n)$  is occupied by  $Inc_1 = (a_1 \vee b_1, a_2 \vee b_2, \dots, a_i \vee b_i, b_{i+1}, \dots, b_n)$ . Now search for the smallest  $j$  such that  $j > i$ ,  $a_j = 1$  and  $b_j = 0$  and when it is found, exchange  $Inc_1 = (a_1 \vee b_1, a_2 \vee b_2, \dots, a_i \vee b_i, b_{i+1}, \dots, b_n)$  with higher order string  $Inc_2 = (a_1 \vee b_1, a_2 \vee b_2, \dots, a_j \vee b_j, b_{j+1}, \dots, b_n)$ . Continue this changes until we have string  $Inc_k = (a_1 \vee b_1, a_2 \vee b_2, \dots, a_n \vee b_n)$  at the desired position of  $(a_1, a_2, \dots, a_n)$ .

**Step 2: decrease the order of the source.** Take string  $(a_1 \vee b_1, a_2 \vee b_2, \dots, a_n \vee b_n)$ , find minimal  $i$ , such that  $a_i = 0$  and  $a_i \vee b_i = 1$  and exchange distance-one strings  $(a_1 \vee b_1, a_2 \vee b_2, \dots, a_n \vee b_n)$  and  $(a_1, a_2, \dots, a_i, a_{i+1} \vee b_{i+1}, \dots, a_n \vee b_n)$ . If the strings  $Dec_1 = (a_1, a_2, \dots, a_i, a_{i+1} \vee b_{i+1}, \dots, a_n \vee b_n)$  and  $(a_1, a_2, \dots, a_n)$  are not equal (otherwise we are done),  $(a_1, a_2, \dots, a_n) \prec Dec_1$  and there exists  $j > i$ , such that  $a_j = 0$  and  $a_j \vee b_j = 1$ . In this case exchange strings  $(a_1, a_2, \dots, a_i, a_{i+1} \vee b_{i+1}, \dots, a_n \vee b_n)$  and  $(a_1, a_2, \dots, a_i, a_{j+1} \vee b_{j+1}, \dots, a_n \vee b_n)$  and call last  $Dec_2$ . Again, in case if  $Dec_2 \neq (a_1, a_2, \dots, a_n)$ , keep decreasing the order by the suggested method until we get  $Dec_s = (a_1, a_2, \dots, a_n)$  and then we are done -  $(a_1, a_2, \dots, a_n)$  is at its place.

Note that in order to bring  $(a_1, a_2, \dots, a_n)$  to its position, we did not modify strings with lower order, so they will stay at their correct places. Second, the number of steps (gates) required to bring any  $(a_1, a_2, \dots, a_n)$  to its correct place equals the sum of “increase order” and “decrease order” steps, which is not more than  $n$ . There are  $2^n$  binary strings, so the method requires at most  $n * 2^n$  steps. ■

**Note,** the constructive proof for this theorem also provides the following statement: any reversible function can be realized in terms of cascades of the gates from set  $Q$ .

Since the functions are reversible, the suggested method can be used in both directions:

- forward: as it is described in the theorem;
- backwards: start with the output part of the truth table, and using the same method, bring it to the first part

(where all the Binary n-tuples are ordered lexicographically). The resulting network in this case will realize the inverse permutation  $f^{-1}$ . But, in order to get a network for the function  $f$ , it is enough to run the obtained network for  $f^{-1}$  in reverse direction.

*Example 3:* We illustrate the proof of the theorem on a (3,3) function  $f$  with the output vector (0, 1, 2, 4, 3, 5, 6, 7). This function was introduced by Miller and Perkowski, and is used in [11] as a benchmark function. Later on, it was named the Miller gate. Here we use the backwards method.

- The first three outputs (0, 0, 0), (0, 0, 1), and (0, 1, 0) are at the correct place.
- Output (1, 0, 0) (shown in it gray) is not in its place. The correct position is occupied by (0, 1, 1) (where the left arrow shows). In order to bring (1, 0, 0) to its place, run steps 1 and 2 from the algorithm.
  - Increase order: interchange (0, 1, 1) with (1, 1, 1) (shown by an arrow from left side).
  - Decrease order: interchange (1, 1, 1) with (1, 0, 1).
  - Decrease order: now we can bring (1, 0, 0) to its place by changing it with (1, 0, 1).

Note that in order to bring (1, 0, 0) to its place we touched strings with the higher order only ((0, 1, 1), (1, 1, 1) and (1, 0, 1)).

- Take the next element in order - (0, 1, 1). It is not in its place, so we color it gray, find its desired place and put an arrow from right pointing the target place.
  - Increase order: interchange (1, 0, 1) with (1, 1, 1) (shown by an arrow from left side).
  - Decrease order: interchange (1, 1, 1) with (0, 1, 1) to put the output string on its place.

Again, no lower order strings were used: (0, 1, 1)  $\prec$  (1, 0, 1)  $\prec$  (1, 1, 1).

- Strings (1, 0, 1), (1, 1, 0) and (1, 1, 1) are at their place, so the network is complete.

The theoretical method uses very wide gates, thus the quantum cost of the resulting circuit is expected to be very high. In addition, it is very likely that the theoretical method produces large networks. To build better circuits than the theoretical algorithm possibly can, we use a different synthesis approach.

## V. HEURISTIC SYNTHESIS

Let  $Q$  be the set of all possible gates with  $n$  inputs. We have shown 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 choose 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, and we call this the distance between two functions. We then choose 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:

TABLE V

TRUTH TABLE OF  $f(x_1, x_2, x_3) = (x_1 \oplus \bar{x}_2x_3, \bar{x}_2, x_1 \oplus x_2x_3)$

$x_1$	$x_2$	$x_3$	$f_1$	$f_2$	$f_3$
0	0	0	0	<b>1</b>	0
0	0	1	<b>1</b>	<b>1</b>	<b>0</b>
0	1	0	0	<b>0</b>	0
0	1	1	0	<b>0</b>	1
1	0	0	1	<b>1</b>	<b>1</b>
1	0	1	<b>0</b>	<b>1</b>	1
1	1	0	1	<b>0</b>	<b>1</b>
1	1	1	1	<b>0</b>	<b>0</b>

TABLE VI

DISTANCE BETWEEN  $f$  AND ITS PARTIAL REALIZATION

$f_1$	$f_2$	$f_3$	$f'_1$	$f'_2$	$f'_3$
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

*Definition 4:* A **partial realization** of  $f$  is any function  $f'$  of the same set of variables.

*Definition 5:* The **distance** between a reversible function  $f$  and its partial realization  $f'$  is the Hamming distance between the output parts of their truth tables (treated as Boolean vectors of length  $n2^n$ ).

*Definition 6:* The **error** of the function  $f$  is its distance to the identity function.

*Example 4:* For the reversible function  $f(x_1, x_2, x_3) = (x_1 \oplus \bar{x}_2x_3, \bar{x}_2, x_1 \oplus x_2x_3)$  whose truth table is shown in Table V the error is equal to 14 (error bits are shown in bold).

*Example 5:* A partial realization  $f'(x_1, x_2, x_3) = (x_1 \oplus \bar{x}_2x_3, x_2, x_3)$  of the reversible function from the previous example is at distance 12 (see Table VI) from the target function  $f$  defined in Example 2.

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

*Lemma 2:* 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:* To see that this is correct, we can 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  $(2^{n-1} - k + s)$  zeros (as  $k$  0 bits are actually ones and  $s$  ones are actually zeros) and  $(2^{n-1} - s + k)$  ones. This observation results in the following set of equations:

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

for which the only solution is  $k = s$ . Therefore, the total number of errors in this column is  $k + s = 2k$ , an even number.

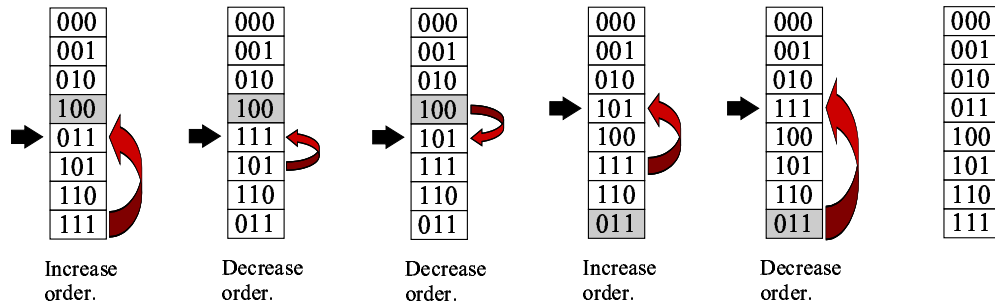


Fig. 6. Building a network.

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 we 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 out of place.

Consider the following simple idea for a synthesis method (it will be used later as a basis for the heuristic synthesis algorithm we try to build): start with the 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  $f$ . However, it is always possible to choose a gate that at least does not increase the distance to  $f$ . For an illustration see Example 6.

*Example 6:* A reversible function with specification  $(x, y) \rightarrow (\bar{y}, \bar{x})$  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 3: (existence of non-positive step)** If the distance between  $f$  and  $f'$  is greater than zero (the partial realization  $f'$  is not the function  $f$  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  $(a_1, a_2, \dots, a_{n-1}, X)$  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  $(a_1, a_2, \dots, a_{n-1}, a_n)$ , where  $a_n = \bar{X}$ . To do so use the gate made of line types  $3 - a_1, 3 - a_2, \dots, 3 - a_{n-1}, 1$  correspondingly, where the Boolean numbers  $a_1, a_2, \dots, a_{n-1}$  are treated as natural numbers. It is easy to see that the gate with this 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 this interchange could happen (Table VII). 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. 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 that each of these steps are non-positive. This fact allows us to formulate the following result and use it as a core to create a synthesis algorithm.

**Theorem 4:** 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 function.

#### A. The Algorithm

The actual implementation of the algorithm works as follows:

- 1) Define the number *MaxMoves* (we used values in the range of 50-500 in our implementation).
- 2) While the distance is greater than zero from among all  $Q$  gates, find the best *MaxMoves* gates. 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 improves (minimizes) the distance between the existing partial realization and the function itself. If such a pair is unique, attach the first gate to the cascade and go back to 2.
- 3) If two or more pairs of gates produce the same improvement to the distance, activate TieBreaker. TieBreaker is the function that finds the third best gate for each pair and if one of the pairs has a better third gate (minimizes the distance function), choose this pair. Then, attach the first gate of the chosen pair to the cascade and go to 2.
- 4) If TieBreaker was not able to find a pair where the third step gives better improvement, then take the pair that gives the best improvement for the first gate. Go to 2.
- 5) If the gate to be assigned is not chosen yet, take the first pair of the gates among those that give the best improvement (from the list produced on step 2). Go to 2.

Theorem 4 states that the distance will not be increased, because there is always a zero step available. In general such a method is not guaranteed to converge, although it does

TABLE VII  
EFFECT OF CHANGING ONE BIT

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

converge for every function we tried. We use this algorithm instead of the theoretical one that is guaranteed to converge, since the latter is likely to give a larger number of steps, because the distance can only decrease by at most two.

An  $(n, n)$  reversible function  $(f_1, f_2, \dots, f_n)$  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, and thusly realize different functions. In our case, for the larger functions (starting from (7,7) functions and larger) we used a heuristic for the output permutation: we took the output permutation that gave the smallest error for the function or its complement. For the functions with a smaller number of variables, we are able to run all the possible permutations and choose the best result.

*Example 7:* Consider the 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: the first step is a zero step, as 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 algorithm terminates in 3 steps). When a function  $(x, y) \rightarrow (\bar{y}, \bar{x})$  with permuted outputs (that is,  $(x, y) \rightarrow (\bar{x}, \bar{y})$ ) can be easily realized with two steps: a subsequent negation of first and second input bits.

### B. Multiple Output Functions

Using the results of Theorem 1 we are able to add the minimal number of garbage bits in order to make a multiple output function reversible. Then, the reversible function specification can be synthesized. However, a better approach would be to calculate the minimal number of garbage outputs, add the corresponding number of bits to the design and assign constant values to them, leaving the garbage outputs unspecified. Then, a gate is added if its addition decreases the value of the error, according to the procedure described above. The benefit in such realization of a multiple output function is that we do not care about some of the outputs: the actual values of the garbage bits are of no interest. This allows us to:

- 1) 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 target outputs only.
- 2) Have a smaller error and therefore, in general, have less steps to make in order to create a network.

### C. Benchmarks

Due to the similarity of gates in the set  $Q$  and the generalized Toffoli gates, we introduce the following notation. The

gate  $Q_{a_1, a_2, \dots, a_n}$  is denoted as  $\text{TOF}(x_{i_1}^{\sigma_1}, x_{i_2}^{\sigma_2}, \dots, x_{i_{s-1}}^{\sigma_{s-1}}; y)$ , which is constructed as follows:

- Write “TOF(”.
- 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  $a_j = 1$ . By the definition of  $Q$  gate such  $j$  will be unique. Finish with the closing bracket “)”.

For example, gate  $Q_{3,1,4,2,4}$  can also be written as  $\text{TOF}(\bar{x}_1, x_4; x_2)$ . This form of writing the gate is better for application use since it makes sense out of the structure of a gate. The old form was used for the simplicity of formulas in mathematical proofs.

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 [8] suggest an approach for reversible cascade synthesis for one output functions. They do not provide experimental results, nor the algorithm, therefore we can not compare those results to ours. Shende *et al.* [17], [18] provide the optimal synthesis method for the (3, 3) reversible functions only. Reversible synthesis of symmetric functions [16] is less general than our approach. Iwama *et al.* [5] base their method on circuit transforms, but they do not provide any experimental data.

We compare our results with three systematic methods: one by Miller [11], another by Mishchenko and Perkowski [14], and a third by Khan and Perkowski [6], [7].

Miller [11] suggests a reversible function synthesis that starts with a reversible specification. He uses spectral techniques to find the best gate (from NOT, CNOT, Toffoli, and Toffoli4, the Toffoli gate with 3 controls) to be added to the network. This method has been modified in [12] to synthesize networks of the presented RCMG model. In his method the output function is required to appear as a set of actual outputs or their negations. Miller also used a postprocessing process to simplify the network (results are given in brackets). The results from all examples in [11] compared to ours are summarized in Table VIII, where **name** is the name of the benchmark function, **in/out** is the number of its inputs/outputs, **Miller** is the number of gates for Miller’s method, and **We** is the number of gates for the proposed synthesis method.

Mishchenko and Perkowski [14] suggest a reversible wave cascade method and evaluate the complexity of some benchmark functions in terms of the size of these cascades. They do not provide the actual design for the described method, but instead they give upper bounds. We compare their results to

TABLE VIII  
COMPARISON WITH MILLER'S RESULTS

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 IX  
COMPARISON WITH MISHCHENKO AND PERKOWSKI'S RESULTS

Function			Garbage		Cost	
name	in	out	MP	We	MP	We
5xp1	7	10	38	0	31	49
9sym	9	1	56	9	52	56
rd53	5	3	19	4	14	13
rd73	7	3	43	6	36	36
xor5	5	1	10	4	5	4

ours and summarize the comparison in Table IX. Although our results are not always better than those of Mishchenko and Perkowski in terms of the total complexity, the important factor, the number of garbage bits, is definitely improved using our approach. We were not able to compare the results for functions with a 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, the first three columns describe the function: the name, number of input bits, and number of output bits of a benchmark function. First pair of columns **MP** and **We** lists the number of garbage outputs from the result of Mishchenko and Perkowski's method and our method; the remaining pair of columns compares the numbers of gates in designs of the benchmark functions for Mishchenko and Perkowski's method and our proposed design respectively. Our method does not always find the realization with the minimum number of gates, but if we consider the cost of a benchmark function to be the sum of the number of gates and the number of garbage outputs, then our method gives a better result.

Results shown by Khan [6], [7] are weaker (although newer) than the results in [14], but for the sake of completeness, we show the comparison in Table X. Note that our results are better in all cases except for *9sym*. However, this is a weakness of the synthesis method we have, not the model, since every single output non-balanced function can be realized with the cost of the number of terms in its minimal EXOR polynomial as is shown in the Section VI of this.

TABLE X  
COMPARISON WITH KHAN'S RESULTS

Function			Garbage		Cost	
name	in	out	Khan	We	Khan	We
5xp1	7	10	63	0	56	49
9sym	9	1	60	9	52	56
rd53	5	3	19	4	17	13
rd73	7	3	47	6	43	36
xor5	5	1	10	4	5	4

It is also interesting to notice that the benchmark *rd53* 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 efficient, compared to the standard non-reversible technological realization of ESOP, the EXOR PLA.

*Example 8:* The  $(5, 1)$ -function *2of5* whose output is 1 if and only if exactly two of the input variables are 1 in terms of ESOP can be realized with 8 terms. Our synthesis method is capable of creating a network (for the proposed structure) with only 7 gates. The function *2of5* is not balanced, therefore the minimal number of garbage outputs 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:  $\text{TOF}(a, \bar{d}, e, \bar{f}; c)$   $\text{TOF}(\bar{b}, c, \bar{d}; f)$   $\text{TOF}(\bar{a}, c, \bar{e}; f)$   $\text{TOF}(\bar{a}, b, d, \bar{e}; f)$   $\text{TOF}(\bar{a}, \bar{f}; e)$   $\text{TOF}(\bar{b}, \bar{c}, d, \bar{e}; f)$   $\text{TOF}(b, \bar{c}, \bar{d}, \bar{e}; f)$ .

It also happens that the synthesis in RCMG model is beneficial to the synthesis of ESOPs, which is shown in the following section.

## VI. RCMG AND ESOP COMPARISON

To exploit the similarity between the two chosen models, note that each of the terms in the ESOP can be treated as a separate gate. All the terms are arranged in the form of a cascade, a string: the EXOR of terms builds the ESOP polynomial. The following summarizes the similarities and differences between the two models.

- Both models use the same operations, namely, AND, EXOR, and negation.
- The gates are similar. Each gate acts as an EXOR of the term built from the input variables. The difference is that in ESOP the set of input variables is not changing while passing through the gates, where for RCMG this is not true.
- The number of distinct gates is comparable:  $n * 3^{n-1}$  for RCMG and  $3^n$  for the ESOP.
- The gates are in a linear order. The terms being "exored" form a string, and generalized Toffoli gates form a cascade. The difference between them is in whether the order matters. The order of terms in a polynomial does not matter, whereas the order of reversible gates in our model does.

*Lemma 4:* By adding a constant input it is possible to use the results of an ESOP minimization to build a reversible network for a single output Boolean function.

*Proof:* Take an  $n$ -input Boolean function and create a zero constant on the input line  $x_{n+1}$ . This may result in non-optimality of the number of garbage outputs. Transform each term  $x_{i_1}^{\sigma_1} x_{i_2}^{\sigma_2} \dots x_{i_k}^{\sigma_k}$  to the gate  $\text{TOF}(x_{i_1}^{\sigma_1}, x_{i_2}^{\sigma_2}, \dots, x_{i_k}^{\sigma_k}; x_{n+1})$ . Such a transformation of each of the terms in the ESOP results in the set of gates of the RCMG; when arranged in a cascade, they form a reversible network for the function. ■

The more interesting question is if the RCMG model is sufficiently efficient in comparison to the ESOP. The answer

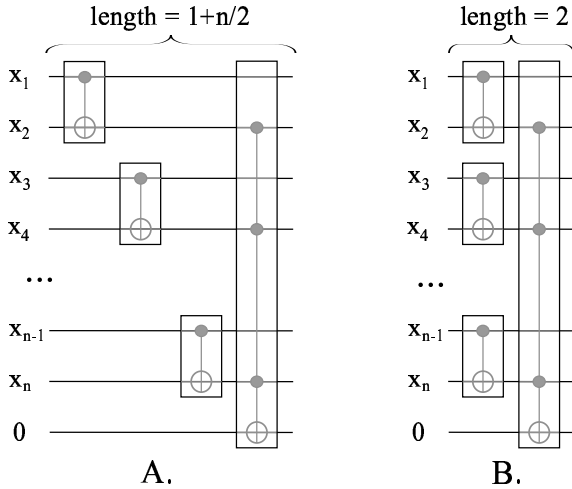


Fig. 7. Two views of the reversible design structure for  $exphard_n$ .

for this question is “yes”, and the following set of Boolean functions grows polynomially for the RCMG and exponentially for the ESOP.

**Definition 7:** For every even integer  $n$ , a Boolean function  $exphard_n(x_1, x_2, \dots, x_n)$  is defined as  $(x_1 \oplus x_2)(x_3 \oplus x_4) \dots (x_{n-1} \oplus x_n)$ .

**Lemma 5:** Function  $exphard_n$  can be realized with cost  $(1 + \frac{n}{2})$  in terms of the RCMG model.

**Proof:** The cascade of gates  $TOF(x_1; x_2) \quad TOF(x_3; x_4) \quad \dots \quad TOF(x_{n-1}; x_n)$   $TOF(x_2, x_4, \dots, x_{n-1}; x_n)$  defines the structure of the network (Fig. 7A). ■

Note that in the actual implementation the first  $n/2$  gates  $TOF(x_1; x_2), TOF(x_3; x_4), \dots, TOF(x_{n-1}; x_n)$  form a single layer. The remaining gate,  $TOF(x_2, x_4, \dots, x_n)$  forms the second layer. Thus, the total length of the network becomes a constant, namely 2 (Fig. 7B).

To show that no ESOP shorter than an ESOP with exponential length can represent function  $exphard_n$ , we need the following Lemmas:

**Lemma 6:** Every term in an optimal ESOP for  $g(x_1, x_2, \dots, x_n, y) = yf(x_1, x_2, \dots, x_n)$ , where  $y \notin \{x_1, x_2, \dots, x_n\}$ , contains variable  $y$  and contains it without negation.

**Proof:** Let  $M$  be an optimal ESOP for the function  $g(x_1, x_2, \dots, x_n, y)$ . Write it as

$$M = yM'_1 \oplus M'_2 \oplus \bar{y}M'_3 = y(M'_1 \oplus M'_3) \oplus (M'_2 \oplus M'_3), \quad (1)$$

where  $M'_1, M'_2$  and  $M'_3$  do not contain  $y$ . The total cost of this ESOP is the sum of the number of terms in  $M'_1, M'_2$  and  $M'_3$ , which is  $|M'_1| + |M'_2| + |M'_3|$ . Let  $N$  be an ESOP for  $f$ . Then  $yN$  forms an ESOP for  $yf$ . Add  $yN$  and  $M$ :

$$0 = yf \oplus yf = yN \oplus M = y(M'_1 \oplus M'_3 \oplus N) \oplus (M'_2 \oplus M'_3).$$

If we write it by components, we have:

$$M'_1 \oplus M'_3 \oplus N = 0, \quad M'_2 \oplus M'_3 = 0. \quad (2)$$

Use this last equality to continue from equation (1):

$$\begin{aligned} yf = M &= y(M'_1 \oplus M'_3) \oplus (M'_2 \oplus M'_3) \\ &= y(M'_1 \oplus M'_3) \oplus 0 \\ &= y(M'_1 \oplus M'_3) \\ &= yM'_1 \oplus yM'_3. \end{aligned}$$

This ESOP has  $|M'_1| + |M'_3|$  terms. Since  $M$  is minimal, the number of terms of  $M'_2$  is zero. Therefore, the number of terms of  $M'_3$  is also zero, which can be seen from the second equation in (2). In other words,  $M = yM'_1$ . ■

**Lemma 7:** Any optimal ESOP for  $g(x_1, x_2, \dots, x_n, y) = yf(x_1, x_2, \dots, x_n)$ , where  $y \notin \{x_1, x_2, \dots, x_n\}$ , has the same complexity as an optimal ESOP for  $f(x_1, x_2, \dots, x_n)$ .

**Proof:** Lemma 6 enables us to factor variable  $y$  out of an optimal ESOP  $M$  for the function  $g(x_1, x_2, \dots, x_n, y)$ :  $M = yM'$ , where  $M'$  is an ESOP that does not contain  $y$  in any form. Let  $y = 1$ . Then,  $M' = (yM')|_{y=1} = M|_{y=1} = (yf(x_1, x_2, \dots, x_n))|_{y=1} = f(x_1, x_2, \dots, x_n)$ . In other words,  $M'$  has the complexity of a minimal ESOP for  $f(x_1, x_2, \dots, x_n)$ , so the ESOP  $M$  does also. ■

**Lemma 8:** A minimal ESOP for the function  $g(x_1, x_2, \dots, x_n, y, z) = yf(x_1, x_2, \dots, x_n) \oplus zf(x_1, x_2, \dots, x_n)$ , where  $y, z$  are variables and  $y, z \notin \{x_1, x_2, \dots, x_n\}$ , consists of at least  $\frac{3|f|}{2}$  terms, where  $|f|$  is the number of terms in a minimal ESOP for  $f$ .

**Proof:** We can take a minimal ESOP  $M$  of the function  $g(x_1, x_2, \dots, x_n, y, z)$  and write it as  $M = yM_1 \oplus M_2 \oplus \bar{y}M_3$ , where  $M_1, M_2$  and  $M_3$  are ESOPs that do not contain the variable  $y$  in either term. Such a decomposition is unique. Notice that the sets of terms in each of  $M_1, M_2$  and  $M_3$  do not intersect:

- $M_1 \cap M_2 = \emptyset$ ;
- $M_1 \cap M_3 = \emptyset$ ;
- $M_2 \cap M_3 = \emptyset$ .

Otherwise, suppose that  $M_1 \cap M_2 \neq \emptyset$ . Then, there exists a term  $t \in (M_1 \cap M_2)$ . Since  $yt \oplus t = \bar{y}t$ , by deleting these two terms from ESOPs  $M_1$  and  $M_2$  and adding it to  $M_3$  we get an ESOP that has complexity (that is, the number of terms in an ESOP) one less than the optimal ESOP  $M$ . This contradicts the optimality of  $M$ . Therefore,  $M_1 \cap M_2 = \emptyset$ . The other two set intersections can be proven to be empty similarly.

Let  $y = 0$  in the ESOP  $M = yM_1 \oplus M_2 \oplus \bar{y}M_3$  for the function  $yf \oplus zf$ . This results in

$$(yf \oplus zf)|_{y=0} = (yM_1 \oplus M_2 \oplus \bar{y}M_3)|_{y=0}$$

and

$$zf = M_2 \oplus M_3. \quad (3)$$

Similarly, assigning  $y = 1$  leads to

$$\bar{z}f = M_1 \oplus M_2. \quad (4)$$

Adding (3) and (4) produces

$$f = M_1 \oplus M_3. \quad (5)$$

Use Lemma 7 to conclude that each of the ESOPs in (3) and (4) has at least  $|f|$  terms. So does the ESOP from (5).

As we proved before, the sets of terms in  $M_1$ ,  $M_2$  and  $M_3$  do not intersect, so based on Lemma 7, (3), (4), and (5), the following system can be written:

$$\begin{cases} |M_2 \oplus M_3| = |M_2| + |M_3| \geq |f| \\ |M_1 \oplus M_2| = |M_1| + |M_2| \geq |f| \\ |M_1 \oplus M_3| = |M_1| + |M_3| \geq |f| \end{cases}.$$

Since

$$|M| = |yM_1 \oplus M_2 \oplus \bar{y}M_3| = |M_1| + |M_2| + |M_3|,$$

the problem of finding the number of terms in a minimal ESOP for  $(yf \oplus zf)$  is bounded by the solution of the following linear optimization problem:

$$\min \begin{cases} |M_2| + |M_3| \geq |f| \\ |M_1| + |M_2| \geq |f| \\ |M_1| + |M_3| \geq |f| \end{cases} \quad (|M_1| + |M_2| + |M_3|) = ?$$

which is given by the expression  $\frac{3|f|}{2}$ . ■

The proof of the following statement would allow us to derive the exact number of terms in a minimal ESOP for  $exphard_n$ .

**Conjecture.** The minimal ESOP for the function  $g(x_1, x_2, \dots, x_n, y, z) = yf(x_1, x_2, \dots, x_n) \oplus zf(x_1, x_2, \dots, x_n)$ , where  $y, z$  are variables and  $y, z \notin \{x_1, x_2, \dots, x_n\}$ , consists of  $2|f|$  terms.

*Theorem 5:* A minimal ESOP for the function  $exphard_n$  has at least  $\sqrt{\frac{3}{2}^n}$  terms.

*Proof:* This result is easily proven by induction using Lemma 8. ■

A better lower bound can be achieved for the best ESOP complexity of the  $exphard_n$  function by saying that every time we apply Lemma 8, the actual ESOP lower bound is  $\left\lceil \frac{3|f|}{2} \right\rceil$  (as a natural number, greater than  $\frac{3|f|}{2}$ ), which brings a larger bound into the next step. The final formula for this observation will look like:

$$|M| \geq \left[ \left[ \left[ \frac{3}{2} \right] * \frac{3}{2} * \dots * \frac{3}{2} \right] \right] \quad (6)$$

Table XI summarizes the results for the function  $exphard_n$ . The first column,  $\mathbf{n}$ , shows the number of inputs. The second column is the number of gates needed for the model RCMG to realize the function. The third column shows the cost for the application of the RCMG model. We used the Exorcism-4 [13], [19] program to calculate the near minimal ESOP for the  $exphard_n$  function. The results of this program are summarized in the fourth column. Note, that this column supports the conjecture. The fifth column shows the theoretically proven lower bound on the minimal ESOP, given by formula (6).

#### A. Multiple Output Functions

One of the reasons that ESOPs are used is their ability to share terms. The RCMG model does not have this property. However, the RCMG model can be united with the mEXOR model introduced in [10] to form a new hybrid model. This will allow the use of multiple EXOR output Toffoli gates with the same control, which is equivalent to the term sharing in the

TABLE XI  
COMPLEXITY OF THE FUNCTION  $exphard_n$

n	RCMG	NRA RCMG	Exorcism-4	ESOP min
2	2	2	2	2
4	3	2	4	3
6	4	2	8	5
8	5	2	16	8
10	6	2	32	12
12	7	2	64	18
14	8	2	128	27
16	9	2	256	41
18	10	2	512	62
20	11	2	1024	93
22	12	2	2048	140
24	13	2	4096	210

non-reversible ESOP model. It can be shown that a quantum realization of such a hybrid gate has a cost that differs from the cost of the original Toffoli gate only marginally. The result of Lemma 4 will now hold for any multiple output Boolean function.

## VII. CONCLUSIONS

In this paper we introduced a synthesis model and a synthesis procedure which allow us to minimize the most important factor of the reversible circuit cost—the number of garbage outputs. We showed that the new gates differ only marginally from the generalized Toffoli gates. We synthesized benchmark functions and achieved good results in comparison to the previously shown results. In some cases our method requires less gates, and in all cases our number of garbage outputs is the theoretically minimal. Finally, synthesis for the RCMG model was shown to be better than synthesis of conventional ESOPs in the sense that no RCMG representation of a function requires more gates than the number of terms in a minimal ESOP. On the other hand, there exists a class of polynomial complexity functions in terms of RCMG which can be realized as an ESOP with exponential cost only.

## REFERENCES

- [1] IBM's test-tube quantum computer makes history. Technical report, [http://researchweb.watson.ibm.com/resources/news/20011219\\_quantum.s.html](http://researchweb.watson.ibm.com/resources/news/20011219_quantum.s.html), Dec. 2001.
- [2] A. Barenco, C. H. Bennett, R. Cleve, D. P. DiVincenzo, N. Margolus, P. Shor, T. Sleator, J. A. Smolin, and H. Weinfurter. Elementary gates for quantum computation. *The American Physical Society*, 52:3457–3467, 1995.
- [3] R. Feynman. Quantum mechanical computers. *Optic News*, 11:11–20, 1985.
- [4] E. Fredkin and T. Toffoli. Conservative logic. *International Journal of Theoretical Physics*, 21:219–253, 1982.
- [5] K. Iwama, Y. Kambayashi, and S. Yamashita. Transformation rules for designing CNOT-based quantum circuits. In *Design Automation Conference*, New Orleans, Louisiana, USA, June 10-14 2002.
- [6] M. H. A. Khan and M. Perkowski. Logic synthesis with cascades of new reversible gate families. In *6th International Symposium on Representations and Methodology of Future Computing Technology (Reed-Muller)*, pages 43–55, March 2003.
- [7] M. H. A. Khan and M. Perkowski. Multi-output ESOP synthesis with cascades of new reversible gate family. In *6th International Symposium on Representations and Methodology of Future Computing Technologies*, pages 144–153, March 2003.
- [8] A. Khlopotine, M. Perkowski, and P. Kerntopf. Reversible logic synthesis by iterative compositions. *International Workshop on Logic Synthesis*, pages 261–266, 2002.

- [9] R. Landauer. Irreversibility and heat generation in the computing process. *IBM J. Research and Development*, 5:183–191, 1961.
- [10] D. Maslov and G. Dueck. Asymptotically optimal regular synthesis of reversible networks. In *International Workshop on Logic Synthesis*, pages 226–231, Laguna Beach, CA, 2003.
- [11] D. M. Miller. Spectral and two-place decomposition techniques in reversible logic. In *Midwest Symposium on Circuits and Systems*, Aug. 2002.
- [12] 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*, pages 56–62, March 2003.
- [13] A. Mishchenko and M. Perkowski. Fast heuristic minimization of exclusive sum-of-products. In *5th International Reed-Muller Workshop*, pages 242–250, August 2001.
- [14] A. Mishchenko and M. Perkowski. Logic synthesis of reversible wave cascades. In *International Workshop on Logic Synthesis*, pages 197–202, June 2002.
- [15] M. Nielsen and I. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, 2000.
- [16] 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 Synthesis*, pages 245–252, 2001.
- [17] V. V. Shende, A. K. Prasad, I. L. Markov, and J. P. Hayes. Reversible logic circuit synthesis. In *International Conference on Computer Aided Design*, pages 125–132, San Jose, California, USA, Nov 10-14 2002.
- [18] V.V. Shende, A.K. Prasad, I.L. Markov, and J.P. Hayes. Synthesis of reversible logic circuits. *IEEE Transactions on CAD*, 22(6):723–729, June 2003.
- [19] N. Song and M. Perkowski. Minimization of exclusive sum of products expressions for multi-output multiple-valued input, incompletely specified functions. *IEEE Transactions on CAD*, 15:385–395, April 1996.
- [20] T. Toffoli. Reversible computing. *Tech memo MIT/LCS/TM-151, MIT Lab for Comp. Sci.*, 1980.



**Dmitri Maslov** received the M.S. in mathematics degree from Lomonosov's Moscow State University, Russia, in 1999, and the M.S. and Ph.D. in computer science from the University of New Brunswick, Canada, in 2002 and 2003, respectively.

He is a postdoctoral fellow in the Department of Computer Science, University of Victoria, Canada. His current research interests include reversible logic and its synthesis, quantum computations, and EXOR minimization.



**Gerhard W. Dueck** (S83-M89) was born in Montevideo, Uruguay. He received the B.Sc., Master, and Ph.D. degrees in computer science for the University of Manitoba, Winnipeg, Manitoba, Canada, in 1983, 1986, and 1988, respectively.

He is currently a professor in the Faculty of Computer at the University of New Brunswick. After completing his PhD he joined St. Francis Xavier University in Antigonish, Nova Scotia. In 1991 he spent a year at the Naval Postgraduate School in Monterey, CA, as a research associate. In 1999 he joined the Faculty of Computer Science at the University of New Brunswick. He has been actively involved in the IEEE Computer Society Technical Committee on Multiple-Valued Logic, where he served as chair in 1998 and 1999. He was program chair of the IEEE International Symposium on Multiple-Valued Logic in 1993 and 2004 and symposium chair in 1997. His research interests include reversible logic, Reed Muller expansions, multiple-valued logic, and digital design.