Garbage in Reversible Designs of Multiple Output Functions

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

Abstract

In this paper we analyze the number of garbage outputs that must be added to a multiple output function to make it reversible. We give the precise formula for the theoretical minimum. For some benchmark functions, we calculate the garbage required by some proposed reversible design methods and compared it to the theoretical minimum. Based on the information about garbage we suggest a new reversible design method, that produces the minimum number of garbage outputs. Finally, we show that our proposed reversible logic structure may have some application in conventional logic design.

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 [7] 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.

Quantum computations are known to solve some exponentially hard problems in polynomial time [10]. 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 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 [3], Toffoli [13], and Fredkin [4] gates. These gates have been studied in detail. However, good synthesis methods have not emerged. Shende *et al.* [12] suggest a synthesis method that produces a minimal circuit with up to 4 input variables. Iwama *et al.* [5] describe transformation rules for CNOT based circuits. These rules may be of use in a synthesis method. Miller [8] 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.* [11] to realize symmetric functions.

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 is more important than the number of gates used, namely the number of garbage out-

*Research supported by the NSERC (CANADA).

puts. 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 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 garbage is illustrated with the following example. Say we want to realize a 5 input 3 output function in a reversible method on a quantum computer, but the design requires 7 additional garbage outputs, resulting in a 10-input 10-output reversible function. In the year 2002 the best quantum computer we have is a 7 qubit computer [1], therefore we will not be able to implement this design. In other words, in case of choosing between increase of the garbage 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 and introduces the model. Since the synthesis of reversible functions differs from the conventional logic synthesis, the following restrictions apply: fan-out and feed back are not permitted [10].

2 Theoretical Minimum

In this paper we define reversible function as follows.

Definition 1. The function $f(x_1, x_2, ..., x_n)$ of n Boolean variables is called reversible if:

1. the number of outputs is equal to the number of inputs;
2. any 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 examples.

Example 1. A 2-input 2-output function given by formula $(x, y) \rightarrow (\bar{x}, x \oplus y)$ is reversible. The correctness of this statement can be verified by looking at the truth table below.

x	y	$\bar{x}$	$x \oplus y$
0	0	1	0
0	1	1	1
1	0	0	1
1	1	0	0

Example 2. A 2-input 1-output function $(x, y) \rightarrow x \oplus y$ is not reversible, since it is not an n-input n-output function. Although, it can easily be made reversible by adding output $\bar{x}$. Note, that for this example we do not need to add an input.

Example 3. Consider the 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 way to make it reversible is to add one input and two outputs so that the function becomes as shown in Table 2. The output vector of the desired function can be observed in the third output column of the table when the value of variable $z = 0$ (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 [13] realizes this function.

x	y	z	x	y	$z \oplus xy$
0	**0**	0	0	0	**0**
0	0	1	0	0	1
0	**1**	0	0	1	**0**
0	1	1	0	1	1
1	**0**	0	1	0	**0**
1	0	1	1	0	1
1	**1**	0	1	1	**1**
1	1	1	1	1	0

Table 1: Reversible function computing the logical AND.

The previous examples show 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-input k-output function ((n,k) function) reversible. ▶

We use the words "constant inputs" to denote the inputs that were 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 input is the constant 0.

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

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

Theorem 1. *For an (n,k) function the minimal garbage to be added to make it reversible is $\lceil \log(M) \rceil$, where M - 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 $(o_1, o_2, ..., o_k)$ has the largest occurrence in the output vector and it appears M times, 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 $(o_1, o_2, ..., o_k)$ had the largest occurrence among all other outputs, all other outputs can be easily separated one from an other by means of $\lceil \log(M) \rceil$ bits. ■

3 Analysis of Garbage in Existing Methods

In this section we analyze garbage in proposed designs. Several design methods (for example [6], [12], and [8]) start with a reversible function. Garbage is introduced during a prepossessing phase, during which the function is made reversible. Note that there are many ways in which the the value of the garbage outputs can be set. Different settings of these variables will lead to results with varying complexity.

Mishchenko and Perkowski [9] suggest a cascade reversible design, called 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 will be $(n + M)$, where n is the number of inputs of multiple output function f and M is the number of cascades (Maitra terms) in the particular realization of a function.

Perkowski *et al.* [11] suggest a regular structure for 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 the garbage of sum of the number of inputs and the number of gates used (additional wires are reserved for the outputs), which gives

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

given the models discussed above, we calculated the number of garbage outputs for some benchmark functions. The following table summarizes the result for methods suggested in [9] and [11] for some benchmark functions used in [9]. The first column shows the name of the function, the second and third are number of input and output bits correspondingly. The fourth column is number of wave cascades, the fifth is the wave cascade method garbage, which is the sum of number of output bits and number from the previous column. The sixth column shows the garbage bits in 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 authors of [11] suggest. In general, such a procedure requires many additional inputs, resulting in a high "garbage price". In cases when the function is not symmetric we put the sign ">" to indicate that the actual garbage is higher. The seventh column is maximal output occurrence, logarithm of which added to the number of function inputs gives the last column, which shows the minimal garbage to make correspondent function reversible.

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

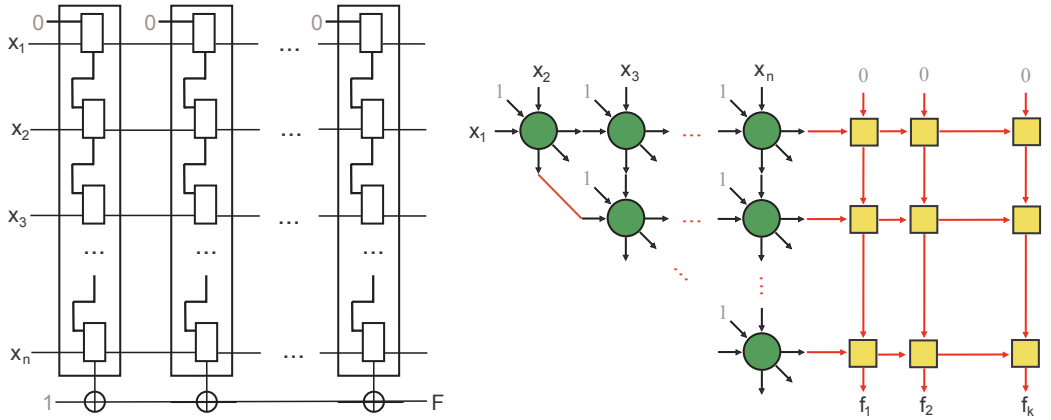

Figure 1: Reversible design methods

name	in	out	VCC	VCG	RPGAG	max out occur	min garbage
5xp1	7	10	31	38	≥ 28	1	0
9sym	9	1	51	60	45	420	9
b12	15	9	28	43	≥ 120	6944	13
clip	9	5	63	72	≥ 45	37	6
in7	26	10	35	61	≥ 351	11651840	24
rd53	5	3	14	19	15	10	4
rd73	7	3	36	43	28	35	6
rd84	8	4	58	66	36	70	7
sao2	10	4	28	38	≥ 55	513	10
t481	16	1	13	29	≥ 136	42016	16
vg2	25	8	184	209	≥ 325	12713984	24

Table 2: Experimental results

4 A new Structure

Many different reversible gates have been introduced in literature. One of the fist gates was the CNOT gate, which capable of producing the "exclusive or" of two input bits as the second output and the first output is equal to the first input. A generalization of CNOT is a 3-input 3-output Toffoli gate. The Toffoli gate negates the third bit iff the first two bits are 1. Fig. 2 shows both gates as they are commonly drawn. In the pictorial representation each of functions, "$\bullet$" denotes the product and "$\oplus$" the "exclusive or".

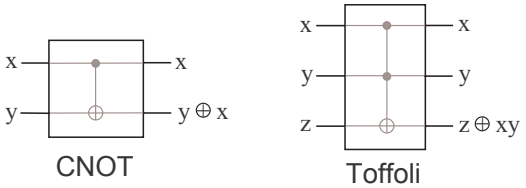

Figure 2: CNOT and Toffoli gates.

Toffoli [13] introduces n-bit input n-bit output $((n,n)$ gate) gate which leaves first $(n-1)$ bits unchanged and negates the last bit if all others are 1. The authors of [5] go further and suggest usage of the generalized (n,n) CNOT (Toffoli) gate, which changes one bit if some of the k bits are 1 (Fig. 3). The changing bit (called the **target**) may also be in any position.

The set of gates that we use in our structure is generalized. We use the same pictorial representation and take (n,n)-gates where each horizontal line is of the following 4 types (Fig. 4):

1. Target line. Each gate should have only one target line appearing at some position j.

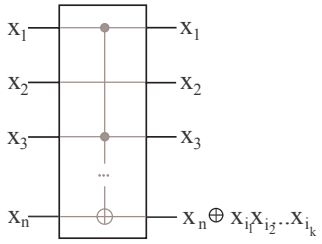

Figure 3: Generalized CNOT gate.

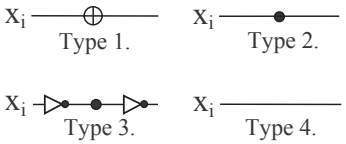

Figure 4: Horizontal line types.

2. Positive control line. If the input on this line is zero, the value of target line will not change. If it is one, look at the remaining positive/negative control lines to determine whether the value on the target line is negated.

3. Negative control line. If the input on this line is one, the value of target line will not change. If it is zero, look at the other remaining positive/negative control lines to 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 $\{x_1, x_2, ..., x_n\}$, 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 numbers $\{\sigma_1, \sigma_2, ..., \sigma_k\}$ 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 $\sigma = 1$ 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.

The graphical representation of a gate is shown in the Fig. 5.

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

Example 4. Take a reversible function $(x_1, x_2, x_3) \rightarrow (x_1 \oplus \bar{x_2} x_3, \bar{x_2}, x_1 \oplus x_2 x_3)$

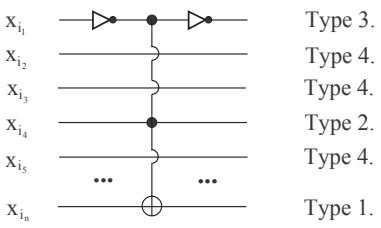

Figure 5: Scheme model

(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 below. A possible implementation is shown in (Fig. 6).

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 3: Truth table

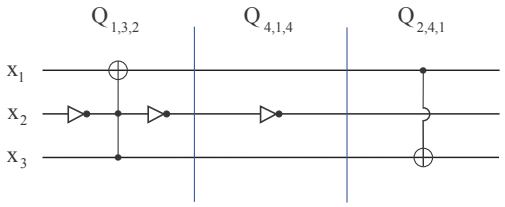

Figure 6: Scheme

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

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

Proof. Let's 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 which there are $(n-1)$ places left to be occupied by positive and negative control and don't care lines to be placed in any combination. The number of ways to put them is, therefore, is 3^{n-1}. This gives the total of $n * 3^{n-1}$ different gates. ■

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

Proof. The number of all reversible functions of n variables is $2^n!$ (as the number of permutations of 2^n elements). The number of different gates is $n3^{n-1}$. Assuming that by taking some of gates and building networks with different orders gives different reversible functions (which is not always true, since, for instance, the gate $Q_{1,4,4,...,4}$, or $\bar{x_1}$ placed two times at a row does nothing) would give us complexity for the hardest function to be $\log_{n3^{n-1}}(2^n!)$. This means that there exists a reversible function which can be realized only with a complexity not less than $log_{n3^{n-1}}(2^n!)$. Using the formula $\ln(k!) = k\ln k - k + \underline{o}(k)$ for $k = 2^n$ write:

$$\log_{n3^{n-1}}(2^n!) = \frac{\ln(2^n!)}{\ln(n3^{n-1})} =$$

$$\frac{2^n\ \ln 2^n - 2^n + \underline{o}(2^n)}{\ln(3^{n-1}) + \ln(n)} = \frac{n2^n - 2^n + \underline{o}(2^n)}{(n-1)\ln 3 + \ln(n)} =$$

$$\frac{(n-1)2^n + \underline{o}(2^n)}{(n-1)\ln 3 + \ln(n)} = \frac{2^n + \underline{o}(2^n/n)}{\ln 3 + \frac{\ln(n)}{n-1}} = \frac{2^n}{\ln 3} + \underline{o}(2^n).$$

QED ■

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

Proof. We use the idea similar to bubble sorting in our constructive proof.

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

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

- strings with less number of ones precede those with larger number of ones;
- strings with equal number of ones are arranged in the lexicographical order.

In other words, the order is as follows: $(0,...,0,0) \prec (0,...,0,1) \prec (0,...,0,1,0) \prec ... \prec (1,0,...,0,0) \prec (0,...,0,1,1) \prec (0,...,0,1,0,1) \prec ... \prec (1,0,...,0,1) \prec ... \prec (1,1,...,1,0) \prec (1,...,1,1)$. We will also use standard order on Boolean constants: $0 \prec 1$

The method is to copy the first part of the truth table to the second, 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 $(a_1, a_2, ..., a_n)$ and bring it to its place. If the string is already at its place, we are done. If it is not, by induction, 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 $(a_1, a_2, ..., a_n)$ is occupied by a $(b_1, b_2, ..., b_n)$. Compose string $(a_1 \vee b_1, a_2 \vee b_2, ..., a_n \vee b_n)$.

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

Step 2: decrease the order of the source. Take string $(a_1 \vee b_1, a_2 \vee b_2, ..., a_n \vee b_n)$, find minimal i, such that $a_i = 0$ and $a_i \vee b_i = 1$ and exchange distance-one strings $(a_1 \vee b_1, a_2 \vee b_2, ..., a_n \vee b_n)$ and $(a_1, a_2, ..., a_i, a_{i+1} \vee b_{i+1}, ..., a_n \vee b_n)$. If the strings $Dec_1 = (a_1, a_2, ..., a_i, a_{i+1} \vee b_{i+1}, ..., a_n \vee b_n)$ and $(a_1, a_2, ..., a_n)$ are not equal (otherwise we are done), $(a_1, a_2, ..., a_n) \prec Dec_1$ and there exists $j > i$, such that $a_j = 0$ and $a_j \vee b_j = 1$. In this case exchange strings $(a_1, a_2, ..., a_i, a_{i+1} \vee b_{i+1}, ..., a_n \vee b_n)$ and $(a_1, a_2, ..., a_i, a_{j+1} \vee b_{j+1}, ..., a_n \vee b_n)$ and call

last Dec_2. Again, in case if $Dec_2 \neq (a_1, a_2, ..., a_n)$, keep decreasing the order by the suggested method until we get $Dec_s = (a_1, a_2, ..., a_n)$ and then we are done - $(a_1, a_2, ..., a_n)$ is at its place.

Note, that in order to bring $(a_1, a_2, ..., a_n)$ to its place we didn't touch strings with lower order, therefore, they will stay at their correct places. Second, the number of steps required to bring any $(a_1, a_2, ..., a_n)$ is correspondent sum $k + s$ which is less than n, since the "increase order" and "decrease order" changes were made at different bits. There are 2^n binary strings, so the method requires at most $n * 2^n$ steps. ■

Note, the constructive proof of 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 inverse permutation f^{-1}. But, in order to get network for the function f it is enough to run obtained network for f^{-1} in reverse direction.

Example 5. Let's illustrate the proof of the theorem on $(3,3)$ function f with the output vector $(0,1,2,4,3,5,6,7)$. This function was introduced by Perkowski and is used in [8] as benchmark function. 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 $(1,0,0)$ (color it gray), which is not in its place, which 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, in order to bring $(1,0,0)$ we touched strings with the higher order only $((0,1,1),(1,1,1)$ and $(1,0,1))$.
- Take the next on the order element - $(0,1,1)$. It is not on 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 built.

In this case the method gave an optimal networks. However, we would not expect this in general, since this method only uses a small subset of the gates available.

5 Non-Reversible Logic Applications

The suggested method is also applicable to non-reversible logic design if the proposed gates are interpreted as correspondent blocks of non-reversible gates NOT, AND and EXOR. The problem of function minimization then becomes a problem of minimization the number of standard NOT, AND and EXOR gates in an implementation. If the final complexity in the suggested method will be less than the one of a PLA, the method may be beneficial. We suggest the following method of eliminating NOT gates from the structure. First, in order to differ non-reversible logic design case from the described in previous sections reversible logic design the "lines" will be called "wires". This makes sense from the point of view of physical implementation. In some designs like the one shown in Fig. 5 two NOT gates may be subsequent. From the point of view of reversible logic we assume we cannot mutually erase them, but in non-reversible case the two subsequent NOT are redundant. For the example from Fig. 5 such a NOT gate pruning gives the design shown in Fig. 8. In general, we divide any gate from the set Q into three logical parts:

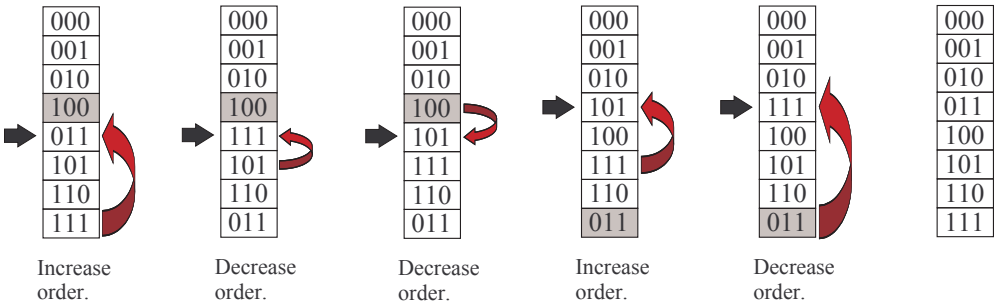

Figure 7: Building a network

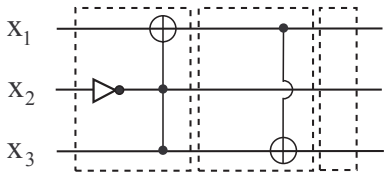

Figure 8: Pruned scheme

- First NOT array: the set of all NOT gates before the vertical line (wire).
- AND-EXOR array: the set of all AND and EXOR gates from the vertical line (wire).
- Last NOT array: the set of all remaining NOT gates.

The general rule of pruning NOT gates is as follows:

1. Define TEMP array - array of NOT gates of the length n, by the definition containing not more then one NOT gate at each place. Initially no NOT gate is presented 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 AND-EXOR and call this structure a block. The last NOT array of gate Q^1 call TEMP. If Q^1 was one of $\bar{x_1}, \bar{x_2}, ..., \bar{x_n}$, add it to TEMP.

3. Take next gate $Q^2 \in Q$ from the network. If $Q^2 \notin \{\bar{x_1}, \bar{x_2}, ..., \bar{x_n}\}$ update TEMP array - "exclusive or" TEMP with first NOT array of Q^2: keep modulo-2 number of sum of number of NOTs at each wire. Unite TEMP array with AND-EXOR array (create a new block), call $Q^2 := Q^1$ and go to 2. If the Q^2 gate was one of gates $\bar{x_1}, \bar{x_2}, ..., \bar{x_n}$, update TEMP array - "exclusive or" TEMP with Q^2, call $Q^2 := Q^1$ and go to 2.

4. When the network is over, create last block - put TEMP array in it.

5. Finally, there might be a case, when a piece of wire consists of gates type 1, 2, 5 and 6. For each of these pieces leave zero or one NOT gate, according to the parity of number of NOT gates seen on the piece. Although such a case is rare, it may give some improvement.

It is easy to see that the network, consisting of described blocks is equivalent to the network built from gates Q. The number of blocks of the pruned networks is the number of gates of initial Q-network minus number of gates from the set $\{\bar{x_1}, \bar{x_2}, ..., \bar{x_n}\}$ of this Q-network plus the TEMP-array. Therefore, both, the set of NOT gates and the length of the structure (number of gates of Q-network) can be only decreased.

The set of all blocks can be described similarly to the set of all Q-gates by naming all the horizontal wires (Fig. 9):

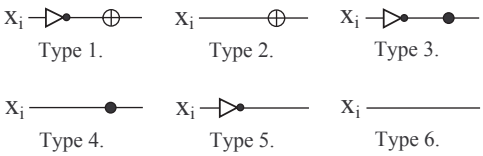

Figure 9: Horizontal wire types.

1. Negative target wire: NOT gate followed by EXOR gate.

2. Positive target wire: EXOR gate only.

3. Negative control wire: NOT followed by AND gate.

4. Positive control wire: AND gate only.

5. Negative don't care wire: NOT gate.

6. Positive don't care wire: no gate.

Where each block contains exactly one wire of type 1 or type 2.

Further simplification of the non-reversible application structure can be done by combining the gates whose sets of input variables do not intersect to form one layer.

6 Conclusion

In the presented work we analyzed an important factor in reversible logic synthesis, the amount of garbage. We suggested the structure to realize reversible functions as cascades of gates from some set Q and proved that for any given multiple output function a reversible network can be built. A theoretical method of the network construction was presented. Although such a design is a costly one, it exists. For a synthesis method see [2].

7 Acknowledgments

The authors wish to acknowledge Dr. Perkowski of Portland State University, USA for his valuable comments.

References

[1] IBM's test-tube quantum computer makes history. Technical report, http://researchweb.watson.ibm.com/resources/news/20011219_quantum.shtml, Dec. 2001.

[2] G. W. Dueck and D. Maslov. Reversible function synthesis with minimum garbage outputs. In *International Symposium on Representations and Methodology of Future Computing Technologies*, March 2003.

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

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

[5] 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.

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

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

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

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

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

[11] 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.

[12] 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.

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