12-2011

Decomposition of Reversible Logic Function Based on Cube-Reordering

Martin Lukac
Tohoku University

Michitaka Kameyama
Tohoku University

Marek Perkowski
Portland State University

Pawel Kerntopf
Warsaw University of Technology

Citation Details

Lukac, Martin, Michitaka Kameyama, Marek Perkowski, and Pawel Kerntopf. "Decomposition of reversible logic function based on cube-reordering." Facta universitatis-series: Electronics and Energetics 24, no. 3 (2011): 403-422.

FACTA UNIVERSITATIS (NIŠ)
SER.: ELEC. ENERG. vol. 24, no. 3, December 2011, 403-422

Decomposition of Reversible Logic Function Based on Cube-Reordering

Martin Lukac, Michitaka Kameyama, Marek Perkowski, and Pawel Kerntopf

Abstract: We present a novel approach to the synthesis of incompletely specified reversible logic functions. The method is based on cube grouping; the first step of the synthesis method analyzes the logic function and generates groupings of same cubes in such a manner that multiple sub-functions are realized by a single Toffoli gate. This process also reorders the function in such a manner that not only groups of similarly defined cubes are joined together but also don't care cubes. The proposed method is verified on standard benchmarks for both reversible and irreversible logic functions. The obtained results show that for functions with a significant portion of don't cares the proposed method outperforms previously proposed synthesis methods.

Keywords: Reversible Logic Synthesis, Toffoli Gates, Cube Reordering, Incompletely Specified Functions

1 Introduction

Reversible Logic Synthesis has been quite actively pursued over approximately the last ten years, mainly due to the fact that quantum computing is naturally a reversible computational system [1] and due to the possibility of building a quantum computer in close future.

Manuscript received Jule 20, 2011. An earlier version of this paper was presented at the Reed Muller 2011 Workshop, May 25-26, 2011, Gustavelund Conference Centre, Tuusula, Finland.

M. Lukac and M. Kameyama are with Graduate School of Information Sciences, Tohoku University, Sendai, Japan (e-mail: lukacm@ecei.tohoku.ac.jp). M. Perkowski is with Department of Computer and Electrical Engineering, Portland State University,Portland, OR, USA (e-mail: mperkows@ee.pdx.edu). P. Kerntopf is with Institute of Computer Science, Warsaw University of Technology, Warsaw, Poland and Department of Theoretical Physics and Informatics, University of Lodz, Lodz, Poland (e-mail: p.kerntopf@ii.pw.edu.pl).

Digital Object Identifier: 10.2298/FUEE1103403L

The reversible logic has been recently gaining some popularity due to the possibility that the quantum computer will be soon realized.

The realization of the quantum computer is motivated by the fact that the current technology is approaching the Moore's limit of transistor integration and the size of a single transistor will soon approach the size of a single elementary particle. At that level, quantum laws are predominant and thus they must be integrated in the design and in the computational paradigms. The reversible logic is well situated to be implemented in quantum technology because quantum technology is naturally reversible. Thus it is reasonable that the reversible logic is increasing in popularity and that more and more synthesis and design methods are being developed.

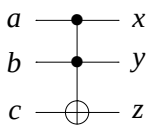
he reversible logic synthesis of incompletely specified functions takes as input a function that contains don't cares in its Karnaugh Map (KMap) and generates a reversible circuit. Up to now few approaches have been proposed for solving problem (as opposed to the fully specified reversible functions). For instance [2] used an ESOP based synthesis where for each cube a single multiple-control Toffoli (MCT) gate was generated. Another approach was proposed more recently [3] where the authors applied BDD (with modified nodes) as the synthesis method of reversible circuits (each node of the BDD was treated as a reversible expansion). Also in [4] several incompletely specified functions are synthesized using an evolutionary automated logic synthesizer.

In this paper we examine a simple heuristic used for the synthesis of reversible logic circuits. In this approach the circuit is analyzed at the level of overlapping control bits allowing to minimize the number of redundant logic gates. This is performed while the circuit is being designed only with Toffoli (CCNOT) gates. In general a function with n inputs require a certain amount of ancilla bits to make the function reversible. Depending on the algorithms the amount of ancilla bits can be either minimal - maximum n - or can be much larger. In the proposed algorithm, the circuits constructed always have at maximum $n-1$ ancilla bits and when compared to methods where more than two control bit Toffoli gates are used the amount of ancilla bits is the same. Finally the proposed method is suitable for such implementations as the Linear Nearest Neighbor (LNN) model of quantum circuits as most of the gates are Toffoli gates acting on neighboring bits.

The paper is organized as follows. Section 2 introduces the basic principles of reversible logic circuits. In Section 3 each step of the proposed synthesis method is introduced and explained. Section 5 describes the experiments and the results. Section 6 concludes this paper.

2 Reversible Logic Basics

Most of developed design methodologies are based on reversible logic gates such as Toffoli or Fredkin gates. The most common is using Toffoli gates [5, 6](Fig. 1) however other less popular approaches using for instance Fredkin gates have been proposed [7, 8].



(a) Pictorial representation of the Toffoli gate

abc	*xyz*
000	000
001	001
010	010
011	011
100	100
101	101
110	111
111	110

(b) Truth table of the Toffoli

abc	*xyz*
000	000
010	010
100	100
110	111
001	001
011	011
101	101
111	110

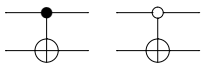
(c) Reordered truth table of the Toffoli gate

Fig. 1. The Toffoli gate

Fig. 1 shows the truth table (Fig. 1(b)) of the Toffoli gate and its pictorial representation (Fig. 1(a)). Observe, that the gate is reversible because the mapping $F : I \rightarrow O$ allows one to compute the inverse mapping $F^{-1} : O \rightarrow I$; the implemented logic is bijective. The reason that the Toffoli gate is popular in the reversible logic synthesis approaches is the fact that it is the universal reversible gate with the smallest number of variables. This can be also seen when the rows of the truth table are reordered properly: the reordering is performed so that the two bits *a* and *b* become the data inputs and the output bit c is the result carrying the output. This can be seen in the Table 1(c). Observe that the reordered Toffoli gate becomes a controlled NAND gate and controlled AND gate (of the *a* and *b* variables) for different values of the *c* control variable (Figure 1(c)). Thus the Toffoli gate can be seen as the direct implementation of the universal irreversible NAND gate, although in fact, it is something more than NAND - it embeds NAND.

The algorithm developed in this paper uses only Toffoli gates and the Feynman gate. The Feynman gate is shown in Fig. 2. The Feynman gates are in two variants; Feynman gate with positive control bit is shown in Fig. 2(a) and Feynman gate with negative control bit is shown in Fig. 2(b) [9].

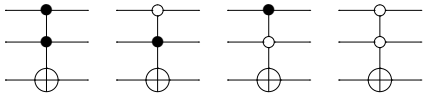
Because the Toffoli gate has two control bits the binary combination of two variables allows to define four different Toffoli gates. These Toffoli gates have controls on the adjacent bits but each type of Toffoli gate has controls according to



(a) (b)

Fig. 2. The two types of Feynman gates used

the possible combination of two bit values. These four types are shown in Fig. 3.



(a) (b) (c) (d)

Fig. 3. The four types of Toffoli gates used

Each Toffoli gate from Fig. 3 is controlled by 11, 01, 10, and 00 respectively.

3 Proposed Cube-Reordering (CR) Algorithm

The algorithm studied in this paper is based on very simple principles:

- Eliminate redundant cubes of Toffoli gates connected in series represented as a SOP logic form
- Synthesize circuits only from two controlled bit Toffoli gates
- Restore bit values only if necessary
- Design circuits in a hierarchical manner

Because the main strategy for minimizing and designing reversible circuits of this algorithm is the re-ordering of the input vectors (cubes) the algorithm is called Cube-Reordering (CR) algorithm. To illustrate the points mentioned above the following example is used.

Example 3.1 Let's look at the definition of the function called 4gt11 (a logical detector of an integer greater than 11 encoded on 4 bits). This function is given in Table 1 which specifies only the *ones* of the function. Assume that the output bit is set to value 0, then only four minterms need to be designed using logic gates to satisfy the specifications. This is because only for outputs of the desired logic function have value 1.

The first observation that can be made is that all four minterms are defined for the same values on the bits *ab* . Thus one can start to create a Toffoli gate and

Table 1. The definition of the function 4gt11.

abcd	f
1100	1
1101	1
1110	1
1111	1

adding a single ancilla bit as shown in Fig. 4. Observe that there are $n-1$ ancilla bits in Fig. 4. As will be seen later, this is the initial configuration of the circuit; for each input bit after the second one one ancilla bit is added. These ancilla bits are used to route the output of the above Toffoli gates if needed. For instance, the first ancilla bit in Fig. 4 is used to represent the *ab* term.

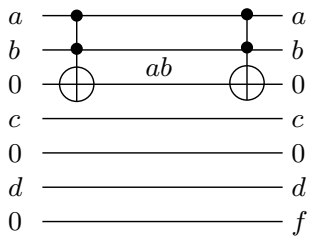


Fig. 4. The first step in the proposed synthesis method.

The next step start by observing that for the input values *ab* there are all four combinations of the cd bits, in particular $\bar{c}\bar{d}$, $\bar{c}d$, $c\bar{d}$ and cd. One can reorder the input minterms with the goal of minimizing the number of required control bits; this means that we order the Toffoli gates defined on the bits *d* and *c* in natural binary order 00, 01, 10 and 11. This is shown in Fig. 5. Later it will be shown how such natural order is used to remove adjacent Toffoli gates.

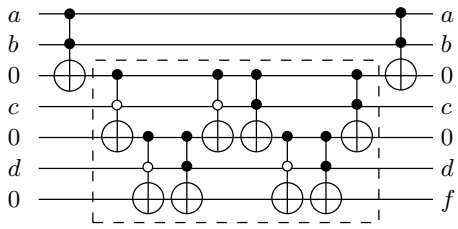


Fig. 5. The second step in the proposed method. All gates are built on the ordered set of all input minterms.

Observe, that the bits *cd* can be replaced by don't cares as they are covering all combinations of values: $00+01+10+11=--$. This can be seen in the dashed

square in Fig. 5. Thus variables *cd* can be skipped and the ancilla bit inserted during the first step becomes the output of the synthesis, as shown in Fig. 6.

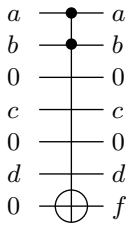


Fig. 6. The circuit of function *4gt11* realized on the bit-array including all initial bits.

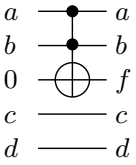


Fig. 7. The final circuit obtained by our method.

This simplified circuit then can be further modified to satisfy some technological constraints (for instance the LNN model) as well as the unused ancilla bits can be removed. The final circuit is shown in Fig. 7. Observe that at this level the cost of the synthesized circuit (5 elementary quantum gates) is lower than the one reported in the RevLib benchmarks (7 gates) [7, 10, 11].

The cost function used in this paper is based solely on the cost of the Toffoli gates; a two-control bit Toffoli gate has a cost of 5 and any Feynman gate has a cost of 1 [10]. In the used cost function any additional gates such as SWAP gates or the single bit NOT gates are not taken into account when calculating the cost.

From Example 3.1 one can observe a multi-level ordering. Because we are restricting the proposed method only to use two-bit controlled Toffoli gates, the ordering is based on a two bit patterns of the input bits. Thus in general for any reversible circuit of an incompletely specified function the method orders all bits in a two by two manner starting on from the top most bits and ending on the lowest ones. The ordering is as described above a natural order but is augmented by the don't cares so that the input cubes follow the order given by $--, -0, -1, 0-, 1-, 00, 01, 10, 11$.

Example 3.2 To understand better the proposed method a more complex example function with eleven input variables and three output variables is shown in Table 2. The format of definition is a PLA format; each column specifies the value of output bits o_0 to o_2 for a cube defined on input bits a to k. This function has in its definition some don't cares that are denoted by *d* in the Table 2.

Before continuing we introduce an encoding of pairs of values of adjacent bits. This means that for every two adjacent bits and starting from the top of the function definition Table 2 we assign a pair of identical values to the adjacent bits in the particular column using the encoding shown in Table 3.

Table 2. The function specification

a	1	*d*	*d*	*d*	*d*	*d*	*d*	*d*	*d*	*d*
b	*d*	1	1	1	1	1	1	1	1	1
c	*d*	*d*	*d*	*d*	*d*	*d*	*d*	*d*	*d*	*d*
d	*d*	0	0	0	0	0	0	0	0	0
e	*d*	0	0	0	0	0	0	0	0	0
f	*d*	0	1	0	0	0	0	0	0	0
g	*d*	0	1	0	0	0	0	0	0	0
h	*d*	1	0	0	1	0	1	0	1	0
i	*d*	1	0	0	1	0	1	0	1	0
j	*d*	0	0	0	1	1	0	0	1	1
k	*d*	1	1	1	1	1	0	0	0	0
o_0	1	1	1	1	0	0	0	0	0	0
o_1	0	0	0	0	1	1	1	1	0	0
o_2	0	0	0	0	0	0	0	0	1	1

Table 3. Encoding of the incompletely defined two-bit cubes. q_t stands for top bit and q_b is the bottom bit for any two adjacent bits in a quantum circuit.

q_t	q_b	encoding
0	0	2
0	1	3
1	0	4
1	1	5
0	d	6
1	d	7
d	0	8
d	1	9
d	d	10

The resulting encoding of Table 2 is shown in Table 4. Observe that the encoding is done two by two bits and thus the encoding is identical for two bits in each pairs ab, cd, ef, gh, ij and k.

Following the steps described in the previous example, one starts by ordering the input vectors (columns) two bits by two bits, from the topmost down to the bottom one. First the two top most bits are ordered, and then recursively the bits below are reordered. The ordering uses natural order of the encoding and thus ultimately the input vectors are ordered in groups, where each group of adjacent input vectors has a common Toffoli gate. Conversely, each adjacent Toffoli gate is placed in natural increasing order of is control input variables.

In other words, ordering each pair of bits creates groups of minterms with common two bits. Then, for each group of input minterms having common upper bits, a reordering is performed two-by-two bits resulting in the Table 5. Observe that in Table 5 the first column is a single group because it has the two topmost bits

Table 4. The function from Table 2 encoded by two bit control types.

a	7	9	9	9	9	9	9	9	9	9
b	7	9	9	9	9	9	9	9	9	9
c	10	8	8	8	8	8	8	8	8	8
d	10	8	8	8	8	8	8	8	8	8
e	10	2	3	2	2	2	2	2	2	2
f	10	2	3	2	2	2	2	2	2	2
g	10	3	4	2	3	2	3	2	3	2
h	10	3	4	2	3	2	3	2	3	2
i	10	4	2	2	5	3	4	2	5	3
j	10	4	2	2	5	3	4	2	5	3
k	d	1	1	1	1	1	0	0	0	0
o_0	1	1	1	1	0	0	0	0	0	0
o_1	0	0	0	0	1	1	1	1	0	0
o_2	0	0	0	0	0	0	0	0	1	1

different from the rest of the input minterms. For the group of minterms with the top most bits encoded by 9, bits e and f creates a two new subgroups with values 2 and 3. Next, for the bits e and f having 2 as encoding, the bits g and h creates two new groups with values of encoding 2 and 3. Recursively applying this approach creates the Table 5.

Table 5. The function from Table 2 ordered and encoded by two bit control types.

a	7	9	9	9	9	9	9	9	9	9
b	7	9	9	9	9	9	9	9	9	9
c	10	8	8	8	8	8	8	8	8	8
d	10	8	8	8	8	8	8	8	8	8
e	10	2	2	2	2	2	2	2	2	3
f	10	2	2	2	2	2	2	2	2	3
g	10	2	2	2	2	3	3	3	3	4
h	10	2	2	2	2	3	3	3	3	4
i	10	2	2	3	3	4	4	5	5	2
j	10	2	2	3	3	4	4	5	5	2
k	d	1	0	1	0	1	0	1	0	1
o_0	1	1	0	0	0	1	0	0	0	1
o_1	0	0	1	1	0	0	1	1	0	0
o_2	0	0	0	0	1	0	0	0	1	0

The next step after the Table 5 has been created is to methodically remove any redundant Toffoli gates. This process creates the so called *activation table*; the table represents the activation of new Toffoli gates. This table is constructed using the pseudo-code shown in Algorithm 1.

The Algorithm 1 traverses the encoded and reordered Table 5 from left to right and from top to bottom. Initially, the *activation table* is initialized to all values

Algorithm 1 Pseudo-Code for the creation of the *Activation Table*

```
 1: Initialize activation Table T to all values be -10
 2: for each input bit I do
 3:    B = 10
 4:    for each input minterm J do
 5:       if b_ij != 10 then
 6:          if b_ij != B then
 7:             B = b_ij
 8:             if B == 0 then
 9:                T_ij = c
10:             else
11:                T_ij = n
12:             end if
13:          else
14:             T_ij = -
15:          end if
16:       end if
17:    end for
18: end for
```

being 10 (don't care) (line 1) and to the size equal to the table defining the original function (in this case 11×10 - we ignore the output bits for the simplicity). Starting in the top left corner of Table 5 (line 2) the temporary variable is set to 10 (line 3). Then traversing each line in the table (variable), each time that the temporary variable B is not equal to the current value - skipping don't cares (line 5) - (indexed by i and j) (line 6), the new value is written to B (line 7). At the same time, in the *activation* table it is written either c (positive control bit) or n (negative control bit). Also observe that when a don't care is encountered nothing is written in the corresponding location of the *activation* table (Table 6) but when the value of at the ij^{th} location in the function table is equivalent to B a don't care sign "-" is written in the activation table (line 14).

The Table 6 is the result of the process described above and in the Algorithm 1. The reason this table is called *activation table* is because each time a letter is written to a location, it represents the fact that a new Toffoli gate has to be inserted - a new gate is activated. The result of this process is a number of letters corresponding to positive and negative control bits of the required Toffoli gates.

The final step is to decide how to place and how to count the Toffoli gates from Table 1. For more complex circuit the proposed method offers two possibilities. Making the assumption (as in [3]) that ancilla bits are available in arbitrary numbers, for each group of cubes one ancilla bit can be introduced into the circuit. To be more precise, each time a Toffoli gate is created an ancilla bit can be inserted

Table 6. The reversible cascade specified by the required control changes - the *activation* table. (Table T)

a	c									
b		c	-	-	-	-	-	-	-	-
c										
d		n	-	-	-	-	-	-	-	-
e		n	-	-	-	-	-	-	-	-
f		n	-	-	-	-	-	-	-	c
g		n	-	-	-	-	-	-	-	c
h		n	-	-	-	c	-	-	-	n
i		n	-	-	-	c	-	-	-	n
j		n	-	c	-	n	-	c	-	n
k		c	n	c	n	c	n	c	n	c
o_0	1	1	0	0	0	1	0	0	0	1
o_1	0	0	1	1	0	0	1	1	0	0
o_2	0	0	0	0	1	0	0	0	1	0

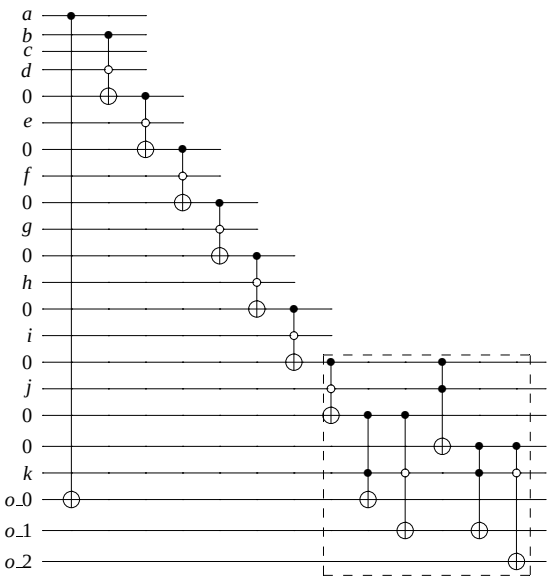


Fig. 8. The five first columns of the resulting circuit created from Table 6 by a direct mapping to Toffoli gates with one ancilla bit per inserted Toffoli gate.

where required. This ancilla bit insertion results in the fact that no Toffoli gate has to be repeated because each Toffoli gate output bit is different. In such case a partial result (the first five columns of Table 6) is shown in Fig. 8.

Continuing in this manner for all groups of two bit until the bottom of the reversible cascade, one obtain a realization of the reversible function that could in

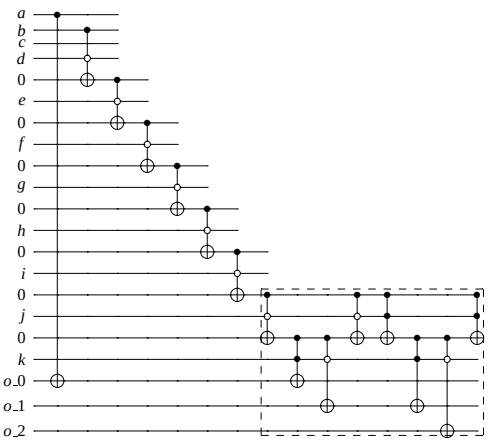


Fig. 9. The five first columns of the resulting circuit created from Table 6 by a direct mapping to Toffoli gates with the restoration of the ancilla bits for later usage.

the worst case require 2^n ancilla bits. This naturally is not acceptable, however for functions that have a high number of don't cares this approach generates results that are less costly in the number of gates as well as in the number of ancilla bits compared with any previously presented approach.

On the other hand, it is possible to design the circuit with a linear number of ancilla bits with a certain number of additional Toffoli gates that have to be inserted. This means that besides each input variable bit we insert an ancilla bit that will be used as the output of a Toffoli gate generating an intermediate result. The additional Toffoli gates are required because these ancilla bits are reused by Toffoli gates in the later stages of the circuit and thus the ancilla bit must be restored to its initial value. This means that in the worst case the number of ancilla bits added is $n-1$. The result of this approach after the first level of cube reordering is shown in Fig. 9. The number of the ancilla bits in this case is equal to $\frac{n}{2}$ but the number of gates is given by $h*2$ (h is the number of 'c' and 'n' elements on the topmost position in every column in Table 6).

To summarize the proposed method can be used to design circuits in the following ways:

1. Design circuit with the number of Toffoli gates equal to $k-h$ where k is the total number of 'c' and 'n' elements in the Table 6. Thus from Table 6 one can design a circuit with exactly $26-1=25$ Toffoli gates. This includes one CNOT gates in the first column. The number of ancilla bits is equal to $k-j$ where j is the number of control bits in the Toffoli gate at the bottom of

each column. Thus the circuit realizing the function from Table 6 requires $10-1=9$ ancilla bits

2. Design circuit with number of gates equal to $(k-h)*2$ and with the number of ancilla bits equal at maximum to the number of input bits; in the case of the Table 6 the maximal number of ancilla bits is thus 11.

4 Cost Reduction and Variable Reordering

Among the several improvements that are possible to the proposed approach, in this section we present two heuristics.

Variable reordering is a well known method in the Boolean function minimization and has been widely used in various aspects including [12–18]. The principle consists of swapping a set of variables in the function. This process often results in more optimal function representation and thus allows a considerable cost reduction in the final function realization. The variable reordering algorithm used in this paper is illustrated by the pseudo-code for algorithm 2.

Algorithm 2 Pseudo-Code for the *Variable Reordering*heuristics

```
 1: configurations = number of input vectors * number of inputs
 2: ordering = new array(sizeof(variables))
 3: Best_Cost = worst_cost
 4: Best_Order = {}
 5: for i = 1 to configurations do
 6:    Cost_i = evaluate(ordering_i)
 7:    if Cost < Best_Cost then
 8:       Best_Cost = Cost_i
 9:       Best_Order = ordering_i
10:    end if
11:    ordering_j = change_ordering(ordering_i)
12: end for
```

Lines 1 and 2 in the pseudo-code 2 initialize the number of tested variable permutations called *configurations* and the array storing the current variable ordering respectively. Line 3 and 4 initialize the *Best_Cost* to a maximum value (always overshooting even the worst possible cost of the circuit) and *Best_Order* to an empty set respectively. The *Best_Cost* variable holds the cost of the circuit calculated from the number of gates and the *Best_Order* variable holds the ordering of the variables corresponding to the best cost. Here the worst possible cost of a given circuit is estimated as the number of control inputs $\times$ number of input vectors; an input vector is each input cube specified in the input file. In this paper the selection of variables to swap is a random process because the best known variable order is not known.

The best possible ordering of the vectors in this case is such that will generate the circuit with minimal number of Toffoli gates, e.g. such ordering that would allow to eliminate as many as possible of Toffoli gates. The random variable ordering however does not always generate the same results and thus in the testing process the evaluation of the variable reordering was performed multiple times in order to confirm the best result.

The cost reduction heuristic is based on the following principles:

1. remove all Toffoli gates that are not required
2. replace adjacent Toffoli gates by CNOT gates whenever possible.

Principle (1) means that as shown in Example 3.2, Toffoli gates used to realize a particular input vector (cube) use a certain amount of the available ancilla bits. Each time a given ancilla bit is required by another following input vector the ancilla bit must be restored to the initial value so it can be re-used.

Principle (2) is used to minimize adjacent Toffoli gates when such gates does not depend on previous variables. For instance, Figure 10 shows where two Toffoli gates can be replaced by one CNOT gate.

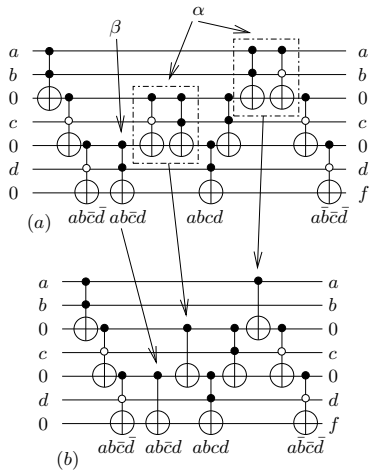


Fig. 10. Example of minimizing Toffoli array by CNOT substitutions

Figure 10 shows a circuit implementing the function $f = abc\bar{d} \oplus ab\bar{c}d \oplus abcd \oplus \bar{a}b\bar{c}\bar{d}$. Figure 10(a) is a direct mapping using only Toffoli gates. Figure 10(b) shows how certain Toffoli gates can be directly replaced by CNOT gates. The Toffoli gates that can be directly replaced are only those that are either on the beginning or at the end of the product term. We call these Toffoli gates the free Toffoli gates. Observe that literals in such terms are also such literals that are the first to change in

the hierarchy of the variables. The gates that are considered to be at the beginning of a product term are the Toffoli gates in the dashed boxes labeled α. The gates considered to be at the end of a product term are the Toffoli gates labeled β.

In more details, the principle (2) can be directly formalized as follows. Let $x_0,\ldots,x_k$ be a set of variables on adjacent k wires; starting form the top wire and ending on the k-th wire. Let a product term be given on the k variables as $x_0 \ldots x_k$ and being created by a set of Toffoli gates as shown in previous examples. Then to invert k-th variable to obtain $x_0 \ldots \bar{x_k}$ one can use the following rule

$$x_0 \ldots \bar{x_k} = x_0 \ldots x_k \oplus x_0 \ldots x_{k-1} \tag{1}$$

Thus in Figure 10 starting by the term $ab\bar{c}\bar{d}$ one can obtain $ab\bar{c}d$ by a single CNOT gate as shown in Figure 10(b) - Toffoli gate labeled β. This particular replacement rule is used as the major cost improvement to the original cube-reordering algorithm. Also observe that because of such rules the original natural order might not always be the best ordering for minimizing the adjacent cubes.

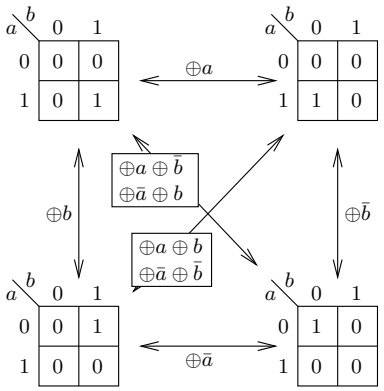


Fig. 11. Diagram showing EXOR relation between the various forms of the Toffoli gates

Figure 11 shows the logical relations that are the base for the Toffoli to CNOT transformations. Observe that a single product term term be moved on a K-map using the EXOR operation. This corresponds to the fact that one can change a Toffoli gate output (product term) to any adjacent product terms by a single CNOT gate. Thus, whenever there are two Toffoli gates that differ in single control bit, the second Toffoli gate can be replaced by a CNOT gate as shown in Figure 10.

The property described in eq. 1 implicitly restrict the usage of the CNOT gates to only the free Toffoli gates. The free Toffoli gates thus represent such gates that can be replaced by single CNOT gates. In particular this means that for two adjacent product terms each built from a set of *m* Toffoli gates and with common *l*

top-most Toffoli gates, the first Toffoli gate that has different one of the control bit can be replaced by a CNOT gate. This is effectively the case of the Toffoli gates labeled α and β in Fig 10. The most important about this property is the fact that in the presented circuit structure the replacement rule removes two Toffoli gates. This is because Toffoli gates with the control bits located on the same bits will generate the output on the same ancilla bit. In such case it is necessary to first restore the ancilla bit - by applying one Toffoli gate - and then apply the next Toffoli gate. In such case, both Toffoli gates are replaced by a CNOT gate.

5 Experiments

Table 7 shows the results of the synthesis method for the benchmarks from the RevLib web-site. The table compares the obtained results to the best reported results in the repository and as can be seen that our algorithm performs quite well for the incompletely specified functions. In fact for the benchmarks that includes functions with a high number of don't cares the algorithm outperforms the best currently reported results.

Because the method currently only supports single output functions, the benchmarks presented include only single output reversible functions. The functions in Table 7 are all such that even if the defined function is defined with more than single output bit, at any given time for any given input only a single output bit is flipped.

The first three columns of the Table 7 are the name, the number of inputs (I) and the number of outputs (O) of the benchmark function respectively. The next column B.P.R.R., is the best previously reported result and the final four columns represent the proposed method and various improvements made to the original CR algorithm. Each result reporting column contains three sub-columns; each representing the number of gates (G), the quantum cost (C) and the number of ancilla bits (A) in the designed circuit, respectively.

The column labeled 'Method 1' represents the results of the CR synthesis algorithm assuming no Toffoli gate repetition for variable restoration. This results in a very large amount of ancilla bits. This corresponds to the approach shown in the Fig. 8. The ancilla bits have not been counted because the method is very wasteful. The results of method 1 are used as the lowest bound on the number of required gates. Method 2 is using the same principle of design as method 1 but the number of Toffoli gates is doubled (each ancilla bit is restored after being used). Using

Table 7. Preliminary results of the proposed methods

F. Name	I	O	B.P.R.R.			Method 1			Method 2			Method 3			Method 4		
			G	C	A	G	C	A	G	C	A	G	C	A	G	C	A
9symml	12	8	129	14193	N.A.	296	1486	N.A.	592	2992	11	577	2885	11	565*6	**2831**	11
add6	12	8	229	**6455**	N.A.	1215+31	6106	N.A	2430+31	12212	11	2297	11485	11	2195+51	11026	11
adr4	12	8	55	**727**	N.A.	212	1060	11	424	2120	11	378	1890	11	290+42	1492	11
alu1	12	8	32	228	N.A.	26+3	133	7	52	266	4	32+3	163	11	19+7	**102**	11
apex5	117	88	2909	**10394**	1025	2526+79	12709	N.A.	5052+79	25418	118	4497+26	22537	118	4409+70	22115	118
apex4	39	3	5376	237963	N.A.	1212+10	6070	N.A.	2024+10	12140	38	2395+10	11975	38	2381+7	**11912**	38
apla	10	12	80	3438	N.A.	74	370	N.A.	148	740	10	106	**530**	10	106	530	10
C17	5	2	9	99	N.A.	5+2	27	0	5+2	27	0	5+2	27	0	4+2	**22**	0
C7552_119	5	16	80	1728	N.A.	44	220	N.A.	88	440	3	84	420	3	56	**280**	3
clip	9	5	174	6731	N.A.	517+5	2590	N.A.	1034+5	5175	8	979	4895	8	798+35	**3980**	8
cm150a	21	1	53	1096	N.A.	64+1	321	N.A.	128+1	641	20	109+1	**546**	20	109+1	546	20
cm151a	19	9	33	888	N.A.	60+1	301	N.A.	120+1	601	18	78	**390**	18	78	390	18
cm152a	11	1	16	252	N.A.	24	120	N.A.	48	240	10	38	190	10	30	**150**	10
cm163a	16	13	39	756	N.A.	61+21	326	N.A.	122+21	631	15	86+29	459	15	38+18	**128**	15
cmb	16	4	18	910	N.A.	47+27	262	N.A.	94+27	497	15	52+2	262	15	43+28	**243**	15
frg1	28	3	212	15265	N.A.	427+2	2137	N.A.	854+2	4170	27	581+2	2907	27	579+3	**2898**	27
frg2	143	139	**3724**	12468	1219	10190+240	51190	N.A.	20380+240	102380+240	142	19485+105	97530	142	19027+334	95469	142
max46	9	1	107	5444	N.A.	253	1265	N.A.	506	2530	8	487	2435	8	419	**2095**	8
misex1	8	7	55	982	N.A.	33+17	182	N.A.	66+17	347	7	50	260	7	32+10	**170**	7
misex3	14	14	1752	199177	N.A.	6355+422	32197	N.A.	12710+422	63972	13	12595	62975	13	8113+281	**40846**	13
mux	21	1	35	**1078**	N.A.	182	910	N.A.	364	1822	20	293	1465	20	286+3	1433	20
pm1	4	10	35	377	N.A.	40	40	3	40	40	3	40	**40**	3	40	40	3
ryy6	16	1	44	4292	N.A.	161	805	N.A.	322	1610	15	281	**1405**	15	281	1405	15
seq	41	35	5990	**19362**	1617	6027+426	30561	N.A.	12054+426	60696	40	11700+1	58501	40	11238+232	56422	40
sqrt8	8	4	40	622	N.A.	60+7	307	N.A.	120+7	607	7	92+2	**462**	7	92+2	462	7
sym9	9	1	28	**108**	12	344	1720	N.A.	688	3440	8	651	3255	8	621+28	3133	8
sym10	10	1	194	25866	N.A.	1800	9000	N.A.	3600	18000	9	3591	**17955**	9	3591	17955	9
t481	16	1	21	**237**	N.A.	1440	7200	N.A.	2880	14400	15	2801	14005	15	2783+9	13924	15
tial	14	8	1041	56203	N.A.	2637+4	13189	N.A.	5274+4	26374	14	5143	25715	14	5031+56	**25211**	14
x2	10	7	38	625	N.A.	29+22	167	N.A.	58+22	312	9	44+8	228	9	22+19	**129**	9

Method 2 the number of the ancilla bits is at maximum equal to the number of input bits because for each Toffoli gate the ancilla bit used is recycled after being used. Method 3 is the optimized method of removal of the redundant Toffoli gates. The method 3 corresponds to the example described in Fig. 9. Finally the column 'Method 4' shows the results obtained when the results of the method 3 have been improved using the variable reordering and the Toffoli to CNOT rewriting rules as shown in Section 4.

From Table 7 it can be seen that our method performs relatively well when used to design circuits with a high number of don't cares. Also, because of the simple representation of the function the method can be used to design circuits with even a very large number of bits. The benchmarks are represented using three different methods.

Observe that as expected the improvements obtained when using the minimization depends on the type of function. This can be seen when looking at the range of the provided cost reduction. This amount can be calculated as the ration $r_r = 1 - \frac{Cost_{method_4}}{Cost_{method_3}}$ with values of $Cost_{method_4}$ and $Cost_{method_3}$ being taken from the sub-columns 3 of columns showing the results of Method 4 and Method 3 respectively. The minimum of improvement is 0 as can be seen for the benchmark function *pm1* or *cm150a* and the maximum is given for the function *x2* with improvement of 0.43. The improvement brought by the reordering of the variables and the gate replacement indeed improved the circuit cost even below the expected minimal circuit cost calculated form the initial variable ordering by method 1.

The observation of these results brings out an interesting point. In fact as it can be expected, the natural ordering of cubes (on which the CR algorithm is based) can be clashing with some of the replacement techniques used to minimize the reversible circuits. For instance, having two adjacent cubes that differ in a single variable might not be the natural order but can be easily minimized using the CNOT substitution rule. Thus an optimal algorithm would require to take all such issues in the scope and the result would indeed be much more minimized circuits.

Finally, it can be pointed out that the method 1 represents also only the upper bound of the complexity of the designed circuits. This can be seen on the circuit shown in Fig. 12. This circuit when designed using the method 1 has 26 gates but when designed using method 2 it only has 29 gates. This is because not all garbage bits are reused and thus do not have to be restored for later usage. This also means that a smart strategy for the removal of the Toffoli gates will lead to circuits that in some cases are much cheaper and in some other case slightly cheaper than the costs shown in Table 7 (Method 3). The improvement brought by the minimization techniques of the circuit from Figure 12 is shown in Figure 13.

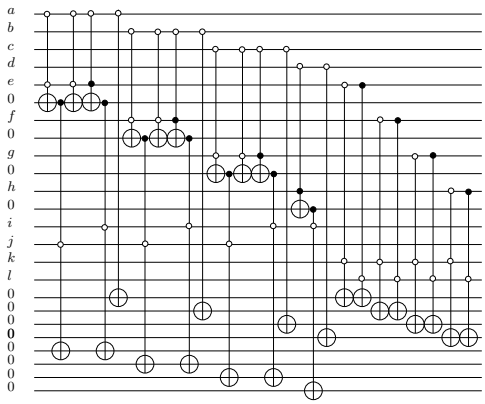


Fig. 12. Example of alu realized by the proposed method with the cost $25*5+4*1=129$ and with four added ancilla bits for intermediary results.

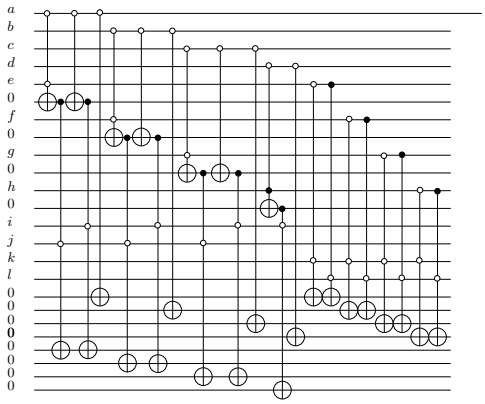


Fig. 13. Alu from Figure 12: circuit cost is minimized by heuristics described in Section 4 resulting in a final cost of $19*5+7*1=102$

6 Conclusion

In this paper we presented an approach to the synthesis of incompletely specified reversible functions. The approach is based on a simple heuristic that regroup similar cubes and order the circuit in a left-to-right and top-to-bottom order. This allows to eliminate redundant Toffoli gates and thus to minimize the circuit cost.

The natural next step in the development of the algorithm is a generalization to multi-output functions. Also the algorithm currently orders only the inputs and thus a more general method of combining the output and input ordering will be also implemented. The current gate counting is still not-optimal. This is because in most of the cases the input vector can be realized in such manner that only certain

ancilla bits must be reused for the realization of a given input. This is due to the fact that not all ancilla bits that are used for the realization of a certain input cube are also used for mutually exclusive input cubes and thus can be used multiple times for a single input vector. In fact as it can be expected an exact method minimizing the circuits to their bare minimum will come close to circuits designed by method 1 (the lower bound of the number of gates) because if the ordering is properly done, in most cases ancilla bits will be used by only mutually exclusive cubes. This is true in the case of the completely defined functions, where the same cubes can be combined and each ancilla bit is used by a single Toffoli gate for each data input. Finally, the long-term future work consists in improving the proposed algorithm by heuristics and more sophisticated methods for Toffoli gates elimination such as template replacement and high level input and output data pattern detection.

Acknowledgments

P. Kerntopf was supported in part by the Polish Ministry of Science and Higher Education under Grant 4180/B/T02/2010/38.

References

[1] R. P. Feynmann, "Quantum mechanical computers," *Optic News*, vol. 11, pp. 11–20, 1985.

[2] K. Fazel, M. Thornton, and J. Rice, "ESOP-based Toffoli gate cascade generation," in *IEEE Pacific Rim Conference on Communications, Computers and Signal Processing*, pp. 206 – 209, 2007.

[3] R. Wille, D. Große, D. Miller, and R. Dreschler, "Equivalence checking of reversible circuits," in *Proceedings of the Inetrnational Symposium on Multi-Valued Logic*, pp. 324–330, 2009.

[4] M. Lukac, *Quantum Logic Synthesis and Inductive Machine Learning*. PhD thesis, Portland State University, 2009.

[5] D. M. Miller, D. Maslov, and G. W. Dueck, "Synthesis of quantum multiple-valued circuits," *Journal of Multiple-Valued Logic and Soft Computing*, vol. 12, no. 5-6, pp. 431–450, 2006.

[6] D. Maslov, G. W. Dueck, and D. M. Miller, "Techniques for the synthesis of reversible Toffoli networks," *ACM Transactions on Design Automation of Electronic Systems*, vol. 12, no. 4, p. 42, 2007.

[7] D. Maslov, G. W. Dueck, and D. M. Miller, "Synthesis of Fredkin-Toffoli reversible networks," *IEEE Transactions on VLSI*, vol. 13, no. 6, pp. 765–769, 2005.

[8] H. Thapliyal and H. Arabnia, "Reversible programmable logic array (RPLA) using fredkin, feynman gates for industrial electronics and applications," in *Proceedings of the International Conference on Computer Design*, pp. 70–76, 2008.

[9] D. Maslov, G. Dueck, D. Miller, and C. Negrevergne, "Quantum circuit simplification and level compaction," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, no. 3, pp. 436–444, 2008.

[10] R. Wille, D. Große, L. Teuber, G. W. Dueck, and R. Drechsler, "RevLib: An online resource for reversible functions and reversible circuits," in *Int'l Symp. on Multi-Valued Logic*, pp. 220–225, 2008. RevLib is available at http://www.revlib.org.

[11] R. Wille and D. Große, "Fast exact Toffoli network synthesis of reversible logic," in *Int'l Conf. on CAD*, pp. 60–64, 2007.

[12] M. Fujita, Y. Matsunaga, and T. Kakuda, "On variable ordering of binary decision diagrams for the application of multi-level logic synthesis," in *Proceedings of the conference on European design automation (EURO-DAC)*, pp. 50–54, 1991.

[13] B. Bollig and I. Wegener, "Improving the variable ordering of OBDDs is NP-complete," *IEEE Transactions on Computers*, vol. 45, no. 9, pp. 993–1002, 1996.

[14] D. Sieling, "The nonapproximability of OBDD minimization," *Information and Computation*, vol. 172, pp. 103–138, 2001.

[15] R. Drechsler, "Evaluation of static variable ordering heuristics for MDD construction," in *Proceedings of the 32nd International Symposium on Multiple-Valued Logic*, pp. 254 – 260, 2002.

[16] O. Grumberg, S. Livne, and S. Markovitch, "Learning to order BDD variables in verification," *Journal of Artificial Intelligence Research*, vol. 18, pp. 83–116, 2003.

[17] D. Miller, D. Y. Feinstein, and M. Thornton, "Qmdd minimization using sifting for variable reordering," *Journal of Multiple-valued Logic and Soft Computing*, pp. 537–552, 2007.

[18] M. Rice and S. Kulhari, "A survey of static variable ordering heuristics for efficient bdd/mdd construction," tech. rep., Bourns College of Engineering, http://www.cs.ucr.edu/ skulhari/StaticHeuristics.pdf, 2008.