DETECTION AND ELIMINATION OF NON-TRIVIAL REVERSIBLE IDENTITIES

Ahmed Younes

Department of Mathematics and Computer Science, Faculty of Science,
Alexandria University, Alexandria, Egypt
ayounes2@yahoo.com

ABSTRACT

Non-Trivial Reversible Identities (NTRIs) are reversible circuits that have equal inputs and outputs. NTRIs of arbitrary size cannot be detected, in general, using optimization algorithms in the literature. Existence of NTRIs in a circuit will cause a slow down by increasing the number of gates and the quantum cost. NTRIs might arise because of an integration of two or more optimal reversible circuits. In this paper, an algorithm that detects and removes NTRIs in polynomial time will be proposed. Experiments that show the bad effect of NTRIs and the enhancement using the proposed algorithm will be presented.

KEYWORDS

Reversible Computing, Quantum Cost, Boolean Functions, Circuit Optimization

1. INTRODUCTION

Reversible logic [1,2] is one of the hot areas of research. It has many applications in quantum computation [3,4], low-power CMOS [5,6] and many more. Synthesis of reversible circuits cannot be done using conventional ways [7]. Synthesis and optimization of Boolean systems on non-standard computers that promise to do computation more powerfully [8] than classical computers, such as quantum computers, is an essential aim in the exploration of the benefits that may be gain from such systems. A function is reversible if it maps an input vector to a unique output vector, and vice versa [9], i.e. we can re-generate the input vector from the output vector (reversibility).

A lot of work has been done trying to find an efficient reversible circuit for an arbitrary reversible function. Reversible truth table can be seen as a permutation matrix of size $2^n \times 2^n$. In one of the research directions, it was shown that the process of synthesizing linear reversible circuits can be reduced to a row reduction problem of $n \times n$ non-singular matrix [10]. Standard row reduction methods such as Gaussian elimination and LU-decomposition have been proposed [11]. In another research direction, search algorithms and template matching tools using reversible gates libraries have been used [12,13,14,15]. These will work efficiently for small circuits. A method is given in [16], where a very useful set of transformations for Boolean quantum circuits is shown. In this method, extra auxiliary bits are used in the construction that will increase the hardware cost. In [17], it was shown that there is a direct correspondence between reversible Boolean operations and certain forms of classical logic known as Reed-Muller expansions. This shows the possibility of handling the problem of synthesis and optimization of reversible Boolean logic within the field of Reed-Muller logic. A lot of work has been done trying to find an efficient

DOI : 10.5121/ijcsea.2012.2406

reversible circuit for an arbitrary multi-output Boolean functions by using templates [18,19] and data-structure-based optimization [20]. A method to generate optimal 4-bit reversible circuits has been proposed [21]. In [22], a very useful set of rules for optimizing reversible circuits and sub-circuits with common-target gates has been proposed. Benchmarks for reversible circuits have been established [23].

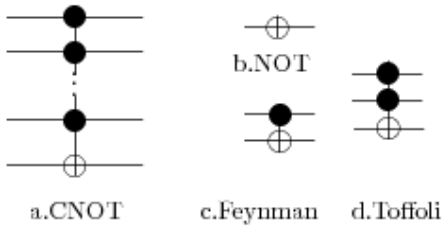

Figure 1. C^nNOT gates. The black circle ● indicates the control bits, and the symbol ⊕ indicates the target bit. (a) C^nNOT gate with n control bits. (b) C^0NOT gate with no control bits. (c) C^1NOT gate with one control bit. (d) C^2NOT gate with two control bits.

The problem of identity circuits has been discussed in the literature, for example [15,18,19,20], where templates for NTRIs has been generated and optimization of the reversible circuits has been done using these templates. The time required to perform the template matching will increase with the number of templates used. In [24,25], a method has been proposed to generate NTRIs, it shows that the size of the NTRIs might increase dramatically and this will make the detection and elimination of NTRIs more complicated.

The aim of the paper is to put a highlight on the problem of non-trivial reversible identities (NTRIs) that might appear after integrating two or more optimal reversible circuits to do more complex tasks. The existence of a NTRI will form a bug in the optimality of the design without affecting the correctness of the circuit. **A NTRT bug** in a computationally correct circuit will cause unnecessary increase in the number of gates and the total quantum cost of the circuit. The paper proposes an algorithm that detects and removes arbitrary NTRIs in polynomial time without affecting the original output of the circuit. The paper is organized as follows. Section 2 gives a short background on reversible gates. Section 3 introduces the problem of NTRIs and gives some examples. Section 4 proposes a polynomial time algorithm to detect and remove NTRIs. Section 5 shows the results of the experiments. The paper ends up with a conclusion in Section 6.

2. BACKGROUND

2.1. Reversible Circuits

In building a reversible circuit with n variables, an $n \times n$ reversible circuit will be used. C^nNOT gate is the main primitive gate that will be used in building the circuit since it was shown to be universal for reversible computation [7]. C^nNOT gate is defined as follows:

Definition 2.1 (C^nNOT *gate*)

C^nNOT is a reversible gate denoted as,

$$C^nNOT(x_{n-2}, x_{n-3}, \cdots, x_0; f), \quad (1)$$

with n inputs: x_{n-2}, x_{n-3},…, x_0 (known as control bits) and f_{in} (known as target bit), and n outputs:

$$\begin{aligned} y_i &= x_i, \; for \; 0 \le i \le n-2, \\ f_{out} &= f_{in} \oplus x_{n-2} x_{n-3} \cdots x_0, \end{aligned} \tag{2}$$

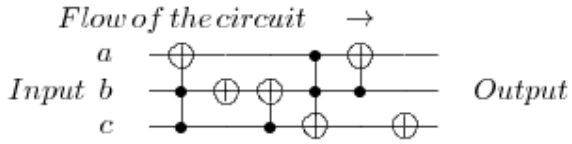

Figure 2. A reversible circuit with three inputs/outputs represented as acyclic directed graph where the flow of the circuit if from left to right.

i.e. the target bit will be flipped if and only if all the control bits are set to 1. Some special cases of the general C^nNOT gate have their own names, C^nNOT gate with no control bits is called *NOT* gate as shown in Figure 1-b, where the bit will be flipped unconditionally. C^nNOT gate with one control bit is called *Feynman* gate as shown in Figure 1-c. C^nNOT gate with two control bits is called *Toffoli* gate as shown in Figure 1-d. For the sake of readability and to keep consistency with the literature, C^0NOT, C^1NOT, C^2NOT and C^3NOT will be written for short as *NOT*, *CNOT*, *TOF* and *TOF4* respectively.

Using the above reversible gates, a reversible circuit is represented as acyclic directed graph (DAG), i.e. loops of gates or internal loops in a gate are not allowed, where the gates are cascaded from left to right. Figure 2 shows a DAG for the circuit *NOT*(*c*) *CNOT*(*b*,*a*) *TOF*(*a*,*b*,*c*) *CNOT*(*c*,*b*) *NOT*(*b*) *TOF*(*c*,*b*,*a*).

2.2. Quantum Cost

Quantum cost is a term that appears in the literature and is used to refer to the technological cost of building C^nNOT gates. The quantum cost of a reversible circuit is subject to optimization as well as the number of C^nNOT gates used in the circuit. The quantum cost of a C^nNOT gate is based primarily on the number of bits involved in the gate, i.e. the number of elementary operations required to build the C^nNOT gate [26]. The calculation of the quantum cost for the circuits shown in this paper is based on the cost table available in [23]. The state-of-art shows that both *NOT*(x_i) and *CNOT*(x_i;f) have quantum cost = 1, *TOF*(x_i,x_j;f) has a quantum cost = 5, and *TOF*4(x_i,x_j,x_k;f) has a quantum cost = 13.

3. NON-TRIVIAL REVERSIBLE IDENTITIES

Circuit identities are usually refer to two or more circuits with the same specification but with different designs [22]. This is usually used to simplify and optimize reversible circuits [19]. In the context of this paper, *reversible identities* will refer to circuits that output their input without any change, i.e. do nothing. The existence of reversible identities in a circuit will increase the number of gates and the quantum cost. Reversible identities can be classified as trivial and non-trivial reversible identities.

Trivial reversible identities are those gates that can be easily detected and removed from a reversible circuit. For example, if two adjacent gates are identical then they can be removed due to reversibility. If the same concept is applied recursively on a circuit, many gates can be

removed. For example, consider the circuit *CNOT*(*b*,*a*) *TOF*(*a*,*b*,*c*) *CNOT*(*c*,*b*) *CNOT*(*c*,*b*) *TOF*(*a*,*b*,*c*) ≡ *CNOT*(*b*,*a*). The two *TOF*(*a*,*b*,*c*) gates cannot be removed together until the two *CNOT*(*c*,*b*) gates are removed. The problem of detection and removal of trivial reversible identities are *not the main target* of the paper, although they will be handled inclusively when dealing with non-trivial reversible identities.

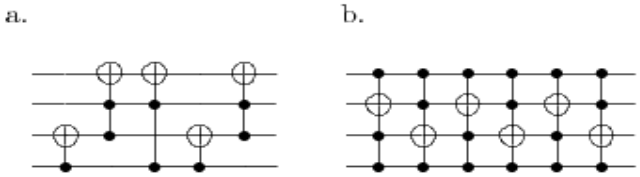

Figure 3. Non-trivial Reversible Identities.

Non-trivial reversible identities (NTRIs) are reversible identities that can be partially detected and removed using template-based optimization algorithm [15,18,19,20,22]. Figure 3 shows two examples of NTRIs. Applying any known optimization algorithm not taking NTRIs in consideration will report that these designs cannot be optimized further. Figure 4-a shows another example of NTRI that is slightly optimized using [22] by decreasing the number of gates by one as shown in Figure 4-b. *NTRIs must be removed completely from the circuit*.

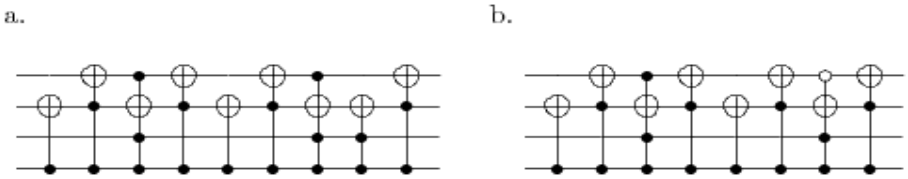

Figure 4. A NTRI before, part-a, and after, part-b, optimization.

It is important here to differentiate between synthesis and optimization of reversible circuits to show a situation where NTRI might arise as a bug. Synthesis of a reversible circuit is the process of constructing, from scratch, the best design for a given specification. Synthesis process always includes optimization. Optimization of a reversible circuit is the process of enhancing an existing design for a given specification by decreasing the number of gates and/or the quantum cost for a given circuit. NTRI is not likely to occur during the synthesizing process, while NTRI might arise as a problem during the optimization process of an integrated circuit that includes two or more computationally correct optimal circuits.

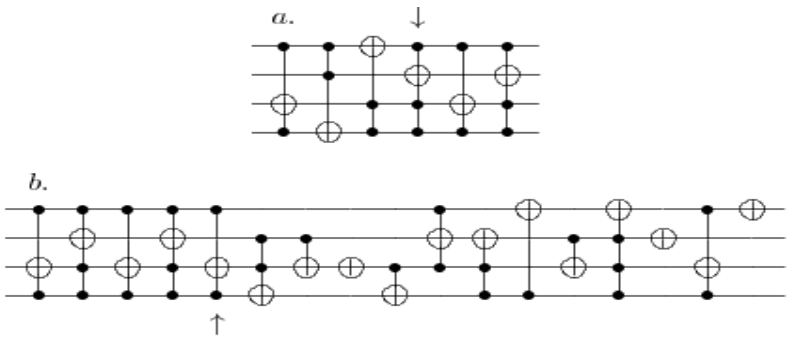

Figure 5. Two Reversible circuits are required to be integrated such that circuit in part-b should be added to the rear of circuit in part-a.

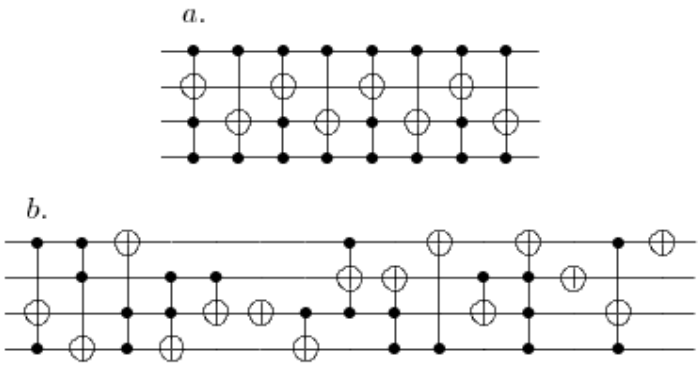

Figure 6. Circuit in part-a shows the discovered NTRI (the bug) and circuit in part-b shows the final circuit after elimination of the NTRI.

To Design a reversible circuit that does a complex task, it will not be practical to do the synthesis process of that circuit from scratch. Instead, the complex task should be broken down to a set of simple modules, design the optimal circuit for each module then integrate the optimal designs in a complete design that perform the required task. NTRI might arise during the process of integration. For example, consider the two reversible circuits shown in Figure 5, where each circuit cannot be optimized further using any known optimization algorithm [22]. Consider that these two circuits should be integrated such that circuit in Figure 5-b should be added to the rear of circuit in Figure 5-a. The integrated circuit contains 23 gates with quantum cost = 125. No further optimization can be done on the integrated circuit although it is bugged by NRTI in the middle. Figure 6-a shows the NRTI and Figure 6-b shows the circuit with 15 gates and quantum cost = 53 after removing the NTRI. Arrows in Figure 5-a and Figure 5-b mark the start and the end of the NTRI respectively. In the next section, an algorithm to remove NTRIs in polynomial time will be proposed. It is important to notice that this algorithm is not a substitution for the current optimization algorithms in the literature; this algorithm is recommended to be used as a module after any optimization algorithm.

4. DETECTION AND ELIMINATION OF NTRIS

4.1. Data Structures

Consider a finite set $A=\{0, 1, ..., N\text{-}1\}$ and a bijection $\sigma : A \to A$, then σ can be written as,

$$\sigma = \begin{pmatrix} 0 & 1 & 2 & \dots & N-1 \\ \sigma(0) & \sigma(1) & \sigma(2) & \dots & \sigma(N-1) \end{pmatrix}, \quad (3)$$

i.e. σ is a permutation of A. Let A be an ordered set, then the top row can be eliminated and σ can be written as,

$$[\sigma(0), \sigma(1), \sigma(2), \dots, \sigma(N-1)]. \quad (4)$$

Any reversible circuit with n inputs can be considered as a permutation σ and Eqn.(4) is the *specification* of this reversible circuit such that $N=2^n$.

In order to detect reversible identities, it is required to store the output of the circuit after applying every gate. Two data structures are used in the proposed algorithm, *Circuit* and *CurrentSpecs*. The first data structure, *Circuit*, is a one dimensional array of size *m*, where *m* is the number of

gates in the reversible circuit. *Circuit* stores the gates of the reversible circuit such that, *Circuit[1]* contains the first gate and *Circuit[m]* contains the last gate.

The second data structure, *CurrentSpecs*, is used to store the output specification after applying every gate in the reversible circuit. *CurrentSpecs* is a two dimensional array of size $N \times m$.

4.2. The Algorithm

The outline of the proposed algorithm is as follows: Run the circuit by applying one gate at a time. Store the specification after applying each gate, and compare the current specification at position *i* with all the previously stored specifications. If specification at position *i* is equal to specification at position *j*, where $j<i$, then remove gates from point *j* to point *i*. Repeat the detection until no more reversible identities are found.

Algorithm 1 Eliminate NTRIs algorithm

```
1:  procedure ELIMINATEIDENTITIES(Circuit)
2:      Finish ← false
3:      while Finish = false do
4:          Finish ← true
5:          CurrentCircuit ← nil
6:          m ← SIZE(Circuit)
7:          for i ← 1, m do
8:              CurrentCircuit ← Circuit[i] ∪ CurrentCircuit
9:              CurrentSpecs[i] ← SPECS(CurrentCircuit)
10:             for j ← 1, i − 1 do
11:                 if CurrentSpecs[j] = CurrentSpecs[i] then
12:                     REMOVEGATES(Circuit, j, i)
13:                     Finish ← false
14:                     j ← i − 1                    ▷ Finish inner loop early
15:                     i ← m                        ▷ Finish outer loop early
16:                 end if
17:             end for
18:         end for
19:     end while
20: end procedure
```

The correctness of the algorithm is proved as follows. The **while loop** from Line:3 to Line:19 will repeat until no more reversible identities exist in *Circuit*. The **for loop** from Line:7 to Line:18 traces *Circuit* one gate at a time and stores the circuit from gate 0 to gate *i* in *CurrentCircuit*. *CurrentSpecs[i]* stores the specification for the circuit at point *i*. The **for loop** from Line:10 to Line:17 checks if there exist any reversible identity by comparing specification of point *i* with specification of point *j* such that $1 \leq j<i$. At Line:11, if an identity is found, then gates from point *j* to point *i* is removed from *Circuit*, and the algorithm starts all over again looking for more reversible identities.

The best case running time exists when no identities exist in *Circuit* where the **while loop** from Line:3 to Line:19 will run once so the algorithm has $\Theta(m^2)$. The worst case running time exists when *Circuit* contains gates such that every two adjacent gates are identical where the **while loop** will repeat $m/2$ times, so the algorithm has $\Theta(m^3)$. The proposed algorithm can detect and eliminate identities from the reversible circuit in polynomial time.

Table 1. Optimal 4-bits reversible circuits from [21] bugged by random NTRIs and optimized using [22], where the pair (*g*,*c*) represents the number of gates (*g*) and the quantum cost (*c*).

Benchmark	Optimal Circuit[21]	Random Identity	Insertion Pt.	Optimal (*g*,*c*)[21]	Bugged (*g*,*c*)	Optimized (*g*,*c*)[22]
4_49	APP1.1.a	APP1.1.b	6	(12,32)	(19,61)	(19,61)
4bit-7-8	APP1.2.a	APP1.2.b	5	(7,19)	(14,40)	(12,34)
decode42	APP1.3.a	APP1.3.b	4	(10,30)	(16,52)	(15,51)
hwb4	APP1.4.a	APP1.4.b	7	(11,39)	(16,64)	(16,62)
imark	APP1.5.a	APP1.5.b	5	(7,19)	(17,43)	(11,37)
mperk	APP1.6.a	APP1.6.b	4	(9,15)	(22,52)	(22,52)
Oc5	APP1.7.a	APP1.7.b	2	(11,39)	(23,65)	(16,52)
Oc6	APP1.8.a	APP1.8.b	11	(12,60)	(20,74)	(20,74)
Oc7	APP1.9.a	APP1.9.a	13	(13,41)	(29,219)	(28,207)
Oc8	APP1.10.a	APP1.10.b	9	(11,47)	(25,197)	(15,79)
primes4	APP1.11.a	APP1.11.b	4	(10,42)	(18,98)	(13,77)
Rd32	APP1.12.a	APP1.12.b	3	(4,8)	(10,54)	(8,46)
shift4	APP1.13.a	APP1.13.b	4	(4,18)	(20,146)	(13,101)

5. EXPERIMENTAL RESULTS

Two software have been developed to test the effect of the existence of arbitrary NTRIs on the optimization of reversible circuits. The first generates random reversible circuits with NTRIs and the second generates random NTRIs. Experiments are based on 4-bits reversible circuits as a prototype.

The first experiment is done by inserting a random NTRI in a randomly chosen insertion point in the middle of optimal 4-bits reversible circuits [21]. Then the latest optimization method [22] using [27] is applied to test the effect of NTRIs on the final number of gates and the quantum cost of the circuit. Table 1 shows the results of the experiment where NTRIs affect the number of gates and the quantum cost. Applying Algorithm 1 before [22] gives the optimal results. Circuits used in the experiment are shown in Appendix 1.

The second experiment is used to test Algorithm 1 by generating random reversible circuits then calculate the number of gates and the quantum cost in the following cases: (1) before applying any method, (2) after applying [22], (3) after applying the proposed method then applying [22]. Table 2 shows the results of the experiment where a better result is obtained using the proposed algorithm. Circuits used in the experiment are shown in Appendix 2.

6. CONCLUSIONS

Non-trivial reversible identities (NTRIs) are bugs that might arise when integrating reversible circuits and will cause a slow down, an increase in the number of gates and the quantum cost of the integrated circuits. Time required by methods based on templates to detect and remove NTRIs will increase with the number of templates used. The structure of NTRIs with large size is not known so far. A polynomial time algorithm is proposed to detect and eliminate NTRIs. The proposed algorithm is not a substitution of any optimization algorithm in the literature. The paper

recommends the proposed algorithm to be used together with any optimization algorithm to handle the problem of NTRIs.

Table 2. Random reversible circuits with NTRIs optimized using [22] alone and optimized using a hybrid system from the proposed algorithm and [22], where the pair (*g*, *c*) represents the number of gates (*g*) and the quantum cost (*c*).

Specification	**Random Circuit**	**Original (*g*,*c*)**	**Optimized (*g*,*c*) [22]**	**Optimized +NTRIs Removal (*g*,*c*)**
[12,7,2,5,0,15,14,11, 6,3,10,1,8,9,4,13]	APP2.1	(21,113)	(17,103)	(10,30)
[7,14,9,6,11,0,13,2, 5,15,10,12,1,4,3,8]	APP2.2	(30,210)	(30,204)	(18,102)
[10,15,0,7,14,9,6,1, 13,12,5,3,11,8,4,2]	APP2.3	(23,103)	(23,101)	(13,43)
[12,9,11,14,6,7,8,10, 2,3,4,5,15,13,0,1]	APP2.4	(22,90)	(22,90)	(9,36)
[0,1,15,8,4,5,9,14, 11,12,7,6,3,13,10,2]	APP2.5	(23,137)	(19,105)	(10,50)
[3,0,1,6,7,2,5,4, 11,8,9,14,15,10,13,12]	APP2.6	(25,133)	(19,99)	(6,14)
[6,11,5,4,2,0,1, 15,14,3,12,8,7,9,13,10]	APP2.7	(21,137)	(20,132)	(15,59)
[12,15,5,8,3,2,1,10, 7,14,13,6,11,0,9,4]	APP2.8	(23,125)	(23,125)	(15,53)
[0,1,6,5,7,8,15,2, 14,13,12,3,11,4,9,10]	APP2.9	(17,65)	(16,64)	(11,47)
[0,10,2,15,8,9,4,1, 6,5,14,3,12,13,11,7]	APP2.10	(20,80)	(19,75)	(13,57)
[8,9,10,2,4,7,6,5, 0,15,13,3,12,14,1,11]	APP2.11	(21,93)	(21,93)	(12,80)
[6,15,0,1,9,2,7,4, 11,10,5,12,3,14,13,8]	APP2.12	(29,73)	(29,73)	(17,53)
[9,3,10,11,12,13,1,7, 0,8,14,2,15,4,5,6]	APP2.13	(25,81)	(17,69)	(12,52)

REFERENCES

[1] C. Bennett. Logical reversibility of computation. IBM J. of Res. and Dev., 17(6):525–532, 1973.

[2] E. Fredkin and T. Toffoli. Conservative logic. International Journal of Theoretical Physics, 21:219–253, 1982.

[3] J. Gruska. Quantum Computing. McGraw-Hill, London, 1999.

[4] M. Nielsen and I. Chuang. Quantum Computation and Quantum Information. Cambridge University Press, Cambridge, United Kingdom, 2000.

[5] A. De Vos, B. Desoete, F. Janiak, and A. Nogawski. Control gates as building blocks for reversible computers. In Proceedings of the 11th International Workshop on Power and Timing Modeling, Optimization and Simulation, pages 9201–9210, 2001.

[6] A. De Vos, B. Desoete, A. Adamski, P. Pietrzak, M. Sibinski, and T. Widerski. Design of reversible logic circuits by means of control gates. In Proceedings of the 10th International Workshop on

Integrated Circuit Design, Power and Timing Modeling, Optimization and Simulation, pages 255-264, 2000.

[7] T. Toffoli. Reversible computing. In W. de Bakker and J. van Leeuwen, editors, Automata, Languages and Programming, page 632. Springer, New York, 1980. Technical Memo MIT/LCS/TM-151, MIT Lab for Computer Science (unpublished).

[8] D. Simon. On the power of quantum computation. In Proceedings of the 35th Annual Symposium on Foundations of Computer Science, pages 116–123, 1994.

[9] V. Shende, A. Prasad, I. Markov, and J. Hayes. Reversible logic circuit synthesis. In Proceedings of ACM/IEEE International Conference on Computer-Aided Design, pages 353–360, 2002.

[10] K. N. Patel, I. L. Markov, and J. P. Hayes. Efficient synthesis of linear reversible circuits. arXiv e-Print quant-ph/0302002, 2003.

[11] T. Beth and M. Roetteler. Quantum algorithms: Applicable algebra and quantum physics. In Quantum Information, pages 96–150. Springer, 2001.

[12] D. Maslov and G. W. Dueck. Reversible cascades with minimal garbage. IEEE Trans. on CAD, 23(11):1497–1509, 2004.

[13] D. Maslov, G. W. Dueck, and D. M. Miller. Fredkin/Toffoli templates for reversible logic synthesis. In Proceedings of the ACM/IEEE International Conference on Computer-Aided Design, page 256, 2003.

[14] D. M. Miller and G. W. Dueck. Spectral techniques for reversible logic synthesis. In Proceedings of the 6th International Symposium on Representations and Methodology of Future Computing Technologies, pages 56–62, 2003.

[15] D. M. Miller, D. Maslov, and G. W. Dueck. A transformation based algorithm for reversible logic synthesis. In Proceedings of the 40th Conference on Design Automation, pages 318–323, 2003.

[16] K. Iwama, Y. Kambayashi, and S. Yamashita. Transformation rules for designing CNOT–based quantum circuits. In Proceedings of the 39th Conference on Design Automation, pages 419–424. ACM Press, 2002.

[17] A. Younes and J. Miller. Representation of Boolean quantum circuits as Reed-Muller expansions. Int. J. of Elect., 91(7):431–444, 2004.

[18] D. Maslov, G. W. Dueck, and D. M. Miller. Simplification of Toffoli networks via templates. In Proceedings of the 16th Symposium on Integrated Circuits and Systems Design, page 53, 2003.

[19] D. Maslov and C. Young and D. M. Miller and G. W. Dueck. Quantum circuit simplification using templates. Design, Automation and Test in Europe, pages 1208-1213, 2005.

[20] A. K. Prasad and V. V. Shende and K. N. Patel and I. L. Markov and J. P. Hayes. Data structures and algorithms for simplifying reversible circuits. J. Emerg. Technol. Comput. Syst., 2(4), October 2006.

[21] O. Golubitsky and S.M. Falconer and D. Maslov. Synthesis of the optimal 4-bit reversible circuits. In Proceedings of the 47th Design Automation Conference, pages 653-656, 2010.

[22] M. Arabzadeh and M. Saeedi and M. S. Zamani. Rule-based optimization of reversible circuits. In Proceedings of The 15th Asia and South Pacific Design Automation Conference (ASPDAC), pages. 849 - 854, 2010.

[23] D. Maslov. Reversible logic synthesis benchmarks. [Online]. Available: http://www.cs.uvic.ca/ dmaslov/.

[24] D. Maslov, G. W. Dueck, and D. M. Miller. Toffoli network synthesis with templates. IEEE Trans. on CAD, 24(6):807–817, 2005.

[25] G. W. Dueck and D. Maslov. Generation of Multiple Control Toffoli Network Templates. In Proceedings of the International Workshop on Logic Synthesis (IWLS), 2007.

[26] 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. Phys. Rev. A, 52(5):3457–3467, Nov 1995.

[27] M. Arabzadeh and M. Saeedi. RCViewer+, version 1.87, 2010, available at http://ceit.aut.ac.ir/QDA/RCV.htm

A Appendix 1

Circuits used in the first experiment where "#" shows the insertion point of the NTRI bug.

APP1.1.a

NOT(a) CNOT(c, a) CNOT(a, d) TOF(a, b, d) CNOT(d, a) # TOF(c, d, b) TOF(a, d, c) TOF(b, c, a) TOF(a, b, d) NOT(a) CNOT(d, b) CNOT(d, c)

APP1.1.b

TOF(b, d, c) CNOT(c, a) CNOT(d, c) CNOT(c, a) TOF4(a, b, d, c) CNOT(d, a) TOF4(a, b, d, c) CNOT(d, c)

APP1.2.a

CNOT(d, b) CNOT(d, a) CNOT(c, d) TOF4(a, b, d, c) # CNOT(c, d) CNOT(d, b) CNOT(d, a)

APP1.2.b

CNOT(d, b) TOF(a, d, c) CNOT(c, a) TOF(a, d, b) CNOT(d, b) CNOT(c, a) TOF(a, d, c) TOF(c, d, b)

APP1.3.a

CNOT(c, b) CNOT(d, a) CNOT(c, a) # TOF(a, d, b) CNOT(b, c) TOF4(a, b, c, d) TOF(b, d, c) CNOT(c, a) CNOT(a, b) NOT(a)

APP1.3.b

TOF(b, d, a) CNOT(d, b) TOF(c, d, b) CNOT(d, b) TOF(c, d, a) TOF(b, d, a) TOF(c, d, b)

APP1.4.a

CNOT(b, d) CNOT(d, a) CNOT(a, c) TOF4(b, c, d, a) CNOT(d, b) CNOT(c, d) TOF(a, c, b) # TOF4(b, c, d, a) CNOT(d, c) CNOT(a, c) CNOT(b, d)

APP1.4.b

TOF(c, d, b) TOF(a, d, c) TOF(a, d, b) TOF(c, d, b) TOF(a, d, c)

APP1.5.a

TOF(c, d, a) TOF(a, b, d) CNOT(d, c) CNOT(b, c) # CNOT(d, a) TOF(a, c, b) NOT(c)

APP1.5.b

TOF(a, d, b) CNOT(c, a) CNOT(d, a) CNOT(c, a) TOF(c, d, b) CNOT(d, a) CNOT(d, b) TOF(c, d, b) CNOT(d, b) TOF(a, d, b)

APP1.6.a

NOT(c) CNOT(d, c) TOF(c, d, b) # TOF(a, c, d) CNOT(b, a) CNOT(d, a) CNOT(c, a) CNOT(a, b) CNOT(b, c)

APP1.6.b

CNOT(d, b) CNOT(d, a) CNOT(c, a) TOF(a, d, b) CNOT(d, b) CNOT(c, a) TOF(a, d, b) TOF(a, d, b) TOF(c, d, a) CNOT(d, a) CNOT(d, b) TOF(a, d, b) TOF(c, d, a)

APP1.7.a

TOF(b, d, c) # TOF(c, d, b) TOF(a, b, c) NOT(a) CNOT(d, b) CNOT(a, c) TOF(b, c, d) CNOT(a, b) CNOT(c, a) CNOT(a, c) TOF4(a, b, d, c)

APP1.7.b

CNOT(d, b) CNOT(c, a) CNOT(d, a) CNOT(c, a) CNOT(d, b) TOF(c, d, a) CNOT(d, a) TOF(c, d, b) CNOT(d, b) TOF(c, d, b) TOF(c, d, a) CNOT(d, b)

APP1.8.a

TOF4(b, c, d, a) TOF4(a, c, d, b) CNOT(d, c) TOF(b, c, d) TOF(c, d, a) TOF4(a, b, d, c) CNOT(b, a) NOT(a) CNOT(c, b) CNOT(d, c) CNOT(a, d) # TOF(b, d, c)

APP1.8.b

TOF(c, d, a) CNOT(c, a) CNOT(d, b) TOF(c, d, a) CNOT(d, a) CNOT(d, a) CNOT(c, a) CNOT(d, b)

APP1.9.a

TOF(b, d, c) TOF(a, b, d) CNOT(b, a) TOF4(a, c, d, b) CNOT(c, b) CNOT(d, c) TOF(a, c, d) NOT(b) NOT(d) CNOT(b, c) TOF(b, d, a) TOF(a, c, d) # CNOT(c, a)

APP1.9.a

TOF4(a, b, d, c) TOF(a, d, b) TOF4(a, b, d, c) TOF4(a, c, d, b) TOF4(a, b, d, c) TOF(a, d, b) TOF4(a, d, c, b) TOF4(a, b, d, c) TOF4(a, d, c, b) TOF(a, d, b) TOF4(a, c, d, b) TOF4(a, b, d, c) TOF(a, d, b) TOF4(a, c, d, b) TOF4(a, b, d, c) TOF4(a, c, d, b)

APP1.10.a

CNOT(d, a) TOF(b, c, a) TOF(c, d, b) TOF4(a, b, d, c) TOF(a, b, d) TOF(a, d, b) NOT(a) NOT(b) TOF(b, d, a) # CNOT(a, d) TOF(b, c, d)

APP1.10.b

TOF(a, d, b) TOF(a, d, b) TOF4(a, c, d, b) TOF4(a, b, d, c) TOF4(a, d, c, b) TOF(a, d, b) TOF4(a, c, d, b) TOF4(a, b, d, c) TOF4(a, b, d, c) TOF4(a, c, d, b) TOF(a, d, b) TOF4(a, c, d, b) TOF4(a, b, d, c) TOF4(a, c, d, b)

APP1.11.a

CNOT(d, c) CNOT(c, a) CNOT(b, c) # NOT(b) TOF(b, c, d) TOF4(a, b, d, c) TOF(a, c, b) NOT(a) TOF4(a, c, d, b) CNOT(b, a)

APP1.11.b

TOF(a, c, b) TOF4(a, c, d, b) TOF(a, d, b) TOF(b, d, a) TOF(b, d, a) TOF(a, d, b) TOF4(a, c, d, b) TOF(a, c, b)

APP1.12.a

TOF(a, b, d) CNOT(a, b) # TOF(b, c, d) CNOT(b, c)

APP1.12.b

TOF(c, d, b) TOF4(a, c, d, b) TOF(a, d, b) TOF4(a, c, d, b) TOF(c, d, b) TOF(a, d, b)

APP1.13.a

TOF4(a, b, c, d) TOF(a, b, c) CNOT(a, b) # NOT(a)

APP1.13.b

TOF(a, d, b) TOF(b, d, c) TOF(a, d, b) TOF(a, d, b) TOF4(a, c, d, b) TOF4(a, b, d, c) TOF(b, d, c) TOF4(a, c, d, b) TOF(b, d, c) TOF(a, d, b) TOF(b, d, c) TOF(a, d, b) TOF4(a, c, d, b) TOF(a, d, b) TOF4(a, c, d, b) TOF4(a, b, d, c)

B Appendix 2

Circuits used in the second experiment where [...] shows the NTRI discovered by the proposed algorithm.

APP2.1

CNOT(b, c) NOT(b) TOF4(a, b, c, d) CNOT(a, b) [TOF(a, d, b) TOF(b, d, c) TOF4(a, c, d, b) TOF4(a, b, d, c) TOF(b, d, c) TOF4(a, c, d, b) TOF(b, d, c) TOF(a, d, b) TOF(b, d, c) TOF4(a, b, d, c)]CNOT(a, d) TOF(a, d, b) NOT(b) TOF(a, b, c) NOT(c) TOF(c, d, b) CNOT(c, d)

APP2.2

TOF4(a, c, d, b) CNOT(c, a) TOF4(a, c, d, b) NOT(c) TOF(b, d, c) TOF(a, d, c) [TOF4(a, c, d, b) TOF(a, d, b) TOF4(a, b, d, c) TOF(a, d, b) TOF4(a, b, d, c) TOF(a, d, b) TOF(a, d, c) TOF4(a, c, d, b) TOF4(a, b, d, c) TOF(a, d, b) TOF(a, d, c) TOF4(a, b, d, c)]TOF(a, b, d) TOF(a, d, b) TOF4(b, c, d, a) CNOT(a, b) CNOT(c, a) CNOT(c, d) TOF(a, d, b) TOF4(a, b, c, d) CNOT(b, c) TOF(a, d, c) TOF4(a, b, c, d) CNOT(b, c)

APP2.3

TOF(a, d, c) TOF4(b, c, d, a) CNOT(d, a) CNOT(c, b) NOT(b) [CNOT(d, b) TOF(b, d, a) TOF(c, d, b) TOF(b, d, a) TOF4(a, c, d, b) CNOT(d, b) TOF(b, d, a) TOF(c, d, b) TOF(b, d, a) TOF4(a, c, d, b) TOF(c, d, b)]CNOT(d, b) CNOT(b, d) CNOT(c, d) TOF4(a, c, d, b) CNOT(a, c) TOF(c, d, b) CNOT(a, b)

APP2.4

NOT(c) NOT(d) NOT(b) CNOT(a, d) [CNOT(d, b) TOF(b, d, a) TOF4(a, c, d, b) TOF(b, d, a) CNOT(d, b) TOF(b, d, a) TOF4(a, c, d, b) TOF(c, d, b) TOF(b, d, a)]CNOT(b, c) CNOT(a, d) TOF(b, c, d) TOF(a, d, b) CNOT(b, c) TOF(c, d, b) TOF4(a, c, d, b) CNOT(b, c) TOF(b, d, a)

APP2.5

TOF(c, d, a) TOF(a, b, c) TOF(a, d, b) CNOT(b, c) [TOF4(a, c, d, b) TOF(c, d, b) TOF(b, d, a) TOF4(a, c, d, b) CNOT(d, b) TOF(c, d, b) TOF(b, d, a) TOF(c, d, b) TOF(b, d, a) TOF4(a, c, d, b) CNOT(d, b) TOF(b, d, a) TOF4(a, c, d, b)]TOF4(a, c, d, b) TOF4(b, c, d, a) CNOT(b, d) CNOT(d, b) CNOT(d, a) TOF(c, d, b)

APP2.6

TOF4(a, c, d, b) CNOT(c, b) TOF4(a, c, d, b) CNOT(a, b) [TOF4(a, c, d, b) TOF(b, d, a) TOF4(a, c, d, b) TOF(b, d, a) CNOT(d, b) TOF(b, d, a) TOF4(a, c, d, b) TOF(b, d, a) CNOT(d, b) TOF4(a, c, d, b) TOF(c, d, b) TOF(b, d, a) TOF(c,

d, b) TOF(b, d, a)]CNOT(c, b) NOT(a) NOT(b) CNOT(c, b) TOF(a, c, b) CNOT(b, c) TOF(a, b, c)

APP2.7

CNOT(a, d) TOF(a, c, b) CNOT(c, d) TOF(a, b, c) CNOT(c, d) CNOT(b, a) TOF4(a, b, c, d) TOF(a, c, d) TOF(c, d, a) NOT(b) CNOT(b, c) [TOF4(a, c, d, b) TOF4(a, b, d, c) TOF4(a, c, d, b) TOF4(a, b, d, c) TOF4(a, c, d, b) TOF4(a, b, d, c)]CNOT(a, c) TOF4(a, b, c, d) TOF(b, d, a) CNOT(d, a)

APP2.8

TOF(a, d, c) TOF(a, b, d) TOF(c, d, a) [TOF4(a, c, d, b) TOF(a, d, c) TOF4(a, c, d, b) TOF(a, d, c) TOF4(a, c, d, b) TOF(a, d, c) TOF4(a, c, d, b) TOF(a, d, c)]TOF(b, c, d) CNOT(b, c) NOT(c) CNOT(c, d) TOF(a, c, b) TOF(c, d, b) CNOT(d, a) CNOT(b, c) TOF4(b, c, d, a) NOT(b) TOF(a, d, c) NOT(a)

APP2.9

TOF(a, c, d) TOF(a, d, b) CNOT(d, b) CNOT(c, a) TOF(a, c, b) TOF(a, b, c) TOF(b, d, c) TOF(a, c, b) TOF4(a, b, c, d) [CNOT(d, c) TOF(b, c, a) TOF(b, d, a) CNOT(d, c) TOF(b, c, a)]CNOT(d, c) TOF(b, c, d) CNOT(b, c)

APP2.10

TOF(a, b, d) CNOT(c, b) TOF(a, b, c) TOF(b, c, d) CNOT(d, c) CNOT(d, b) TOF(b, c, d) CNOT(a, b) TOF(c, d, b) TOF(a, c, d) [CNOT(d, c) TOF(b, c, a) CNOT(d, c) TOF(b, c, a) TOF(b, d, a)]CNOT(a, d) TOF(c, d, a) TOF(b, d, a) TOF(c, d, a) TOF4(b, c, d, a)

APP2.11

TOF(a, c, b) NOT(c) TOF4(b, c, d, a) TOF(a, b, c) CNOT(a, c) TOF4(a, c, d, b) TOF4(a, b, c, d) [CNOT(d, c) CNOT(c, b) TOF(b, c, a) CNOT(c, a) CNOT(c, b) TOF(b, c, a) CNOT(d, c)]TOF(b, d, a) CNOT(a, c) TOF4(a, c, d, b) CNOT(c, d) TOF(a, b, c) TOF(a, b, d) NOT(c)

APP2.12

TOF(c, d, a) TOF(a, c, d) CNOT(c, d) NOT(a) TOF(c, d, a) [CNOT(d, c) CNOT(c, b) TOF(b, c, a) CNOT(c, b) CNOT(d, c) CNOT(c, a) CNOT(d, c) CNOT(c, a) CNOT(d, c) CNOT(c, a) TOF(b, c, a) TOF(b, d, a)]TOF(a, c, d) NOT(d) CNOT(d, c) TOF(a, c, d) NOT(a) NOT(b) CNOT(a, d) TOF4(a, c, d, b) NOT(c) CNOT(c, a) CNOT(b, c) TOF(a, b, d)

APP2.13

CNOT(c, a) TOF(b, d, a) NOT(c) TOF(a, c, d) TOF(a, b, c) TOF4(a, b, d, c) TOF(b, d, a) CNOT(d, a) CNOT(b, d)[CNOT(d, c) TOF(b, d, a) CNOT(d, c) TOF(b, c, a) CNOT(d, c) TOF(b, c, a) CNOT(d, c)]NOT(a) NOT(d) TOF(a, b,d) TOF(b, c, a) NOT(d) TOF(a, d, b) NOT(c) NOT(d) TOF(b, c, d)

Author

Ahmed Younes is presently employed as Computer Science Assistant Professor at Alexandria University. He obtained his PhD from Birmingham University (UK). He published papers in Quantum Algorithms, Quantum cryptography and Reversible Circuits.