An Optimized Majority Logic Synthesis Methodology for Quantum-Dot Cellular Automata

Article *in* IEEE Transactions on Nanotechnology · April 2010

DOI: 10.1109/TNANO.2009.2028609 · Source: IEEE Xplore

3 authors:

Kun Kong
University of International Business and Economics
3 PUBLICATIONS **93** CITATIONS
SEE PROFILE

Yun Shang
Chinese Academy of Sciences
35 PUBLICATIONS **181** CITATIONS
SEE PROFILE

Ruqian lu
Chinese Academy of Sciences
124 PUBLICATIONS **548** CITATIONS
SEE PROFILE

An Optimized Majority Logic Synthesis Methodology for Quantum-Dot Cellular Automata

Kun Kong, Yun Shang, and Ruqian Lu

***Abstract*—Quantum-dot cellular automata (QCA) has been widely considered as a replacement candidate for complementary metal-oxide semiconductor (CMOS). The fundamental logic device in QCA is the majority gate. In this paper, we propose an efficient methodology for majority logic synthesis of arbitrary Boolean functions. We prove that our method provides a minimal majority expression and an optimal QCA layout for any given three-variable Boolean function. In order to obtain high-quality decomposed Boolean networks, we introduce a new decomposition scheme that can decompose all Boolean networks efficiently. Furthermore, our method removes all the redundancies that are produced in the process of converting a decomposed network into a majority network. In existing methods, however, these redundancies are not considered. We have built a majority logic synthesis tool based on our method and several existing logic synthesis tools. Experiments with 40 multiple-output benchmarks indicate that, compared to existing methods, 37 benchmarks are optimized by our method, up to 31.6%, 78.2%, 75.5%, and 83.3% reduction in level count, gate count, gate input count, and inverter count, respectively, is possible with the average being 4.7%, 14.5%, 13.3%, and 26.4%, respectively. We have also implemented the QCA layouts of 10 benchmarks by using our method. Results indicate that, compared to existing methods, up to 33.3%, 76.7%, and 75.5% reduction in delay, cell count, and area, respectively, is possible with the average being 8.1%, 28.9%, and 29.0%, respectively.**

***Index Terms*—Majority networks, minimum cost, quantum-dot cellular automata (QCA), redundancy removal.**

I. Introduction

In THE past few decades, complementary metal-oxide semiconductor (CMOS) technology has consistently played a vital role in implementing high-density, high-speed and low-power very large scale integrated systems. It provided the required dimensional scaling to support the integration. However, several studies have predicted that this technology is approaching its fundamental physical limits [1]. Some new nanotechnologies, such as quantum-dot cellular automata (QCA) [2]–[5], single electron tunneling (SET) [6], [7], and tunneling phase logic (TPL) [8], have been proposed as possible replacement candidates for CMOS. In this paper, we mainly consider QCA which is one of the top emerging technologies with potential application in future computers.

QCA provides a new method of computation and information transformation. In QCA, binary information is encoded by the configuration of electrical charges in a QCA cell. Computation is realized via the Coulombic interaction between neighboring cells. Because of the Coulombic nature of quantum cells, current does not flow between cells. Moreover, power dissipation in QCA circuits is ultra low compared with conventional CMOS circuits [2], [3].

In traditional CMOS design, logic reduction methods always convert Boolean functions into simplified sum of products (SOP) or product of sums (POS) expressions, and logic circuits are implemented using AND and OR gates based on these SOP or POS formulae. However, in QCA design, logic circuits are based on majority gates. Due to the complexity of multi-level majority circuits, there is no efficient way to convert SOP or POS formulae into majority expressions. Thus, traditional logic synthesis methods in CMOS can not efficiently produce QCA circuits based on majority gates. Hence, an efficient technique is required to convert Boolean functions into majority expressions. That is, we need automatic majority logic synthesis for QCA.

Research in majority logic synthesis dates back to 1960s [9]–[11]. Akers [9], Miller and Winder [10], and Muroga [11] employed logic synthesis methods based on reduced-unitized-table, Karnaugh-map (K-map), and Shannon's decomposition principles, respectively. However, these methods have a common drawback that they are suitable only for synthesizing small networks by hand. Recently, some majority logic reduction methods targeting QCA circuits have been proposed in [12]–[15]. However, these methods can process only three-variable Boolean functions. In order to synthesize arbitrary multi-variable Boolean functions, two comprehensive QCA majority synthesis methodologies have been introduced in [16] and [17].

In majority logic synthesis, there still exist some important aspects that are not solved or considered by the existing methods [9]–[17]. First, we are unaware of any work that systematically discusses the cost of a majority expression, that is, the cost of the QCA implementation corresponding to this majority expression, which plays a crucial role in implementations of QCA circuits, just as the cost of an SOP (POS) formula performs an important function in implementing traditional CMOS circuits. Second, in any one of the existing comprehensive majority logic synthesis methodologies [16], [17], all Boolean networks are

Manuscript received December 24, 2008. First published July 28, 2009; current version published March 10, 2010. This work was supported by the National Natural Science Foundation of China (NSFC) Major Research Program under Grant 60496324, NSFC projects under Grants 60736011, 60603002, and 90818026, 863 projects under Grant 2007AA01Z325, 973 projects under Grant 2009CB32070. The review of this paper was arranged by Associate Editor D. Hammerstrom.

K. Kong and Y. Shang are with the Institute of Mathematics, Academy of Mathematics and Systems Science, Chinese Academy of Sciences, Beijing 100190, China.

R. Lu is with the Institute of Mathematics, Academy of Mathematics and Systems Science, Chinese Academy of Sciences, Beijing 100190, China, and also with the Department of Computer Science and Engineering, Shanghai Key Lab of Intelligent Information Processing, Fudan University, Shanghai 200433, China. He is also with the Key Lab of Intelligent Information Processing, Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190, China (e-mail: rqlu@math.ac.cn).

Digital Object Identifier 10.1109/TNANO.2009.2028609

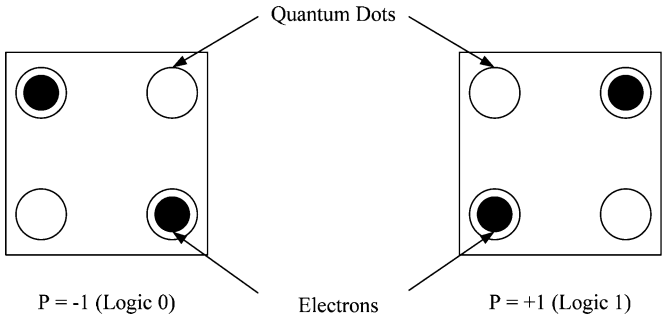

Fig. 1. QCA cell and its binary behavior.

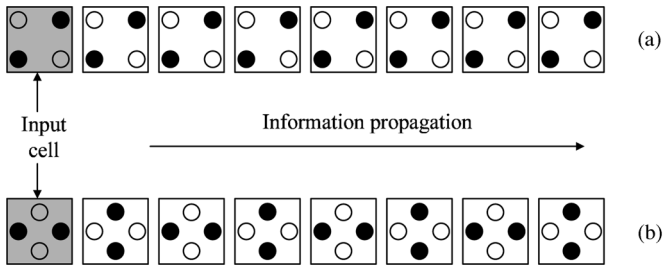

Fig. 2. (a) 90° QCA wire. (b) 45° QCA wire.

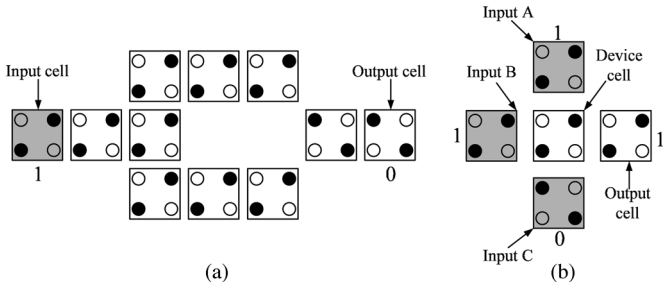

Fig. 3. Two basic QCA devices. (a) Inverter. (b) Majority gate.

decomposed by only one decomposition method, which results in many complex decomposed networks. If we use some other methods, these networks will become much simple. Third, some redundant nodes are produced when converting a decomposed network into a majority network. These redundancies cause unnecessary hardware requirements. To our best knowledge, no work has been done to solve this problem.

In this paper, we present an optimized methodology for comprehensive majority logic synthesis. The novel contributions of our work are as follows.

1) By using our method, we can obtain a minimal majority expression for an arbitrary three-variable Boolean function. Therefore, our QCA layout for this function is optimal.
2) We offer a new decomposition scheme to decompose all Boolean networks. Compared to the existing method [16], up to 69.0% reduction in node count is obtained and the average reduction is 15.67%.
3) Our method removes all the redundancies that are not considered in existing methods. Experiments show that an average reduction of 5.8% in majority gate count can be obtained by this removal.

The rest of this paper is organized as follows. In Section II, we introduce some background material of QCA technology and logic synthesis. In Section III, we describe our methodology in detail. In Section IV, we discuss experimental results and comparisons. A conclusion is addressed in the last section.

II. Background Material

In this section, we introduce some background material to help the reader understand the remainder of this paper better.

A. QCA Basics

A quantum cell consists of four quantum dots located at the corners of a square and two extra electrons that can tunnel between the dots. Electrostatic repulsion causes the extra electrons to occupy diagonally opposite sites. These two electron configurations are denoted as cell polarization $P = +1$ and $P = -1$, respectively. Binary information is encoded by using $P = +1$ to represent logic "1" and $P = -1$ to represent logic "0". Fig. 1 shows a QCA cell and its two electron configurations.

B. QCA Logic Devices

The fundamental QCA logic devices include a QCA wire, QCA inverter, and QCA majority gate.

A QCA wire is just a line of QCA cells. The wire is driven at the input cell by a cell with a fixed/held polarization [18]. The propagations in a 90° QCA wire and 45° QCA wire are shown in Fig. 2(a) and (b), respectively.

Cells placed diagonally to each other have reverse polarizations. This characteristic is exploited to implement an inverter, such as the one shown in Fig. 3(a).

The logic function of a majority gate is

$$M(a, b, c) = ab + ac + bc$$

where a, b, and c are arbitrary inputs. The tendency of the majority device cell to move to a ground state ensures that it assumes the polarization of the majority of the three input cells, as shown in Fig. 3(b). By setting one input permanently to logic "1" or logic "0", an OR gate or AND gate can be obtained, respectively, that is

$$M(a, 1, b) = a + b$$
$$M(a, 0, b) = ab.$$

C. QCA Clock

A QCA clock consists of four phases which are called Switch, Hold, Release, and Relax [3], [5], as shown in Fig. 4. During the Switch phase, the interdot barriers are slowly raised and the QCA cells become polarized according to the state of their drivers (that is, their input cells). During the Hold phase, the interdot barriers are kept high and the QCA cells retain their states. In the Release phase, the barriers are lowered and the cells are allowed to relax to an unpolarized state. Finally, in

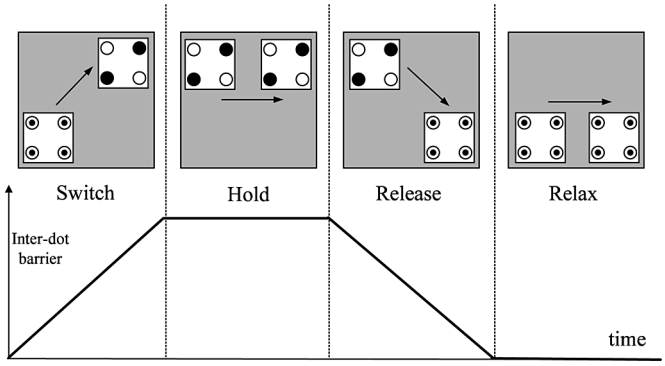

Fig. 4. Four phases of a QCA clock.

the Relax phase, the barriers are kept low and the cells remain unpolarized.

A QCA circuit is partitioned into serial zones. These zones can be of irregular shape, but their size must be within certain limits imposed by fabrication and dissipation concerns [19]. All cells in the same zone are controlled by the common clock signal. The scheme of clock zones permits an array of QCA cells to make a certain calculation and then have its state frozen, and finally, have its output serve as the input to the next clock zone.

D. Algebraic and Boolean Expressions

A SOP form is considered as a set of cubes and a cube is a set of literals. For instance, $F = ab + ac + ad$ is considered a set of three cubes, i.e., $F = \{C_1, C_2, C_3\}$, where each cube is a set of two literals: $C_1 = \{a, b\}, C_2 = \{a, c\}$, and $C_3 = \{a, d\}$. An algebraic expression $F = \{C_i\}$ is one in which no cube contains another (i.e., $\forall i, j, i \neq j, C_i \not\subseteq C_j$) and no cube contains the form xx or xx'. An expression that is not algebraic is Boolean [20], [21].

E. Algebraic and Boolean Factored Forms

A factored form is defined recursively by three rules: 1) a product is either a single literal or a product of factored forms; 2) a sum is either a single literal or a sum of factored forms; and 3) a factored form is either a product or a sum. A factored form F is said to be algebraic if the SOP expression obtained by multiplying F out directly (without using $xx' = 0$ and $xx = x$, and single-cube containment) is algebraic. F is a Boolean factored form if it is not algebraic [20], [21].

F. r-Feasible Functions and Networks

If a Boolean function has at most *r* input variables, it is called *r*-feasible. Otherwise, it is called r-infeasible. A Boolean network is *r*-feasible if every node in this network is *r*-feasible [16], [22].

III. Methodology and Implementation

In this section, we introduce our majority logic synthesis methodology and its implementation details.

A. Overview of the Method

Fig. 5 gives a flow diagram of the main steps in our methodology. The input to our methodology is an arbitrary multiple-output Boolean function F. Its output is a functionally equivalent majority network G.

The procedure begins by preprocessing function F. During preprocessing, F is checked to determine if it is correct. If it is, we simplify F and factor it algebraically. Otherwise, we show error information and end the procedure. Next, F is decomposed into a network in which each node has at most three input variables. Then, each node in the decomposed network is checked to determine if it is a majority function. If it is, we convert it into the corresponding majority expression directly. Otherwise, we convert it into a minimal majority expression by our method. Finally, after all nodes in the decomposed network have been synthesized, we check that if there are redundant nodes in the network. If this is the case, we remove these redundant nodes and update the network for the next redundancy judgement. Otherwise, we terminate the procedure.

The above description of methodology can be summarized as the following four steps.

1) *Preprocessing*. Error detection, Boolean simplification and algebraic factorization are executed to the input function. This step provide a good starting point for decomposition.
2) *Decomposing*. The algebraic factored form is decomposed into a Boolean network in which every node has at most three input variables.
3) *Converting*. We convert the function of every node in the decomposed network into its minimal majority expression form.
4) *Removing*. all the redundancies produced after conversion are removed.

B. Preprocessing

We first debug the given multiple-output Boolean function. After making sure that the function is correct, we simplify it and factor it algebraically. Then, we obtain an algebraic factored multiple-output form which provides a good starting point for majority network synthesis.

In our methodology, we do preprocessing by making use of simplification and factorization methods in sequential interactive synthesis (SIS) [22]. Fig. 6 gives a script consisting of a list of commands in SIS. An algebraic factored form is obtained by executing the script.

We present a simple example to demonstrate this process. Consider the function $F = ai'k' + bi'k + ck'i + dki + cdi$. First, F is simplified to $F = ai'k' + bi'k + ck'i + dki$. Then, F is factored algebraically to $F = (ak' + bk)i' + (ck' + dk)i$.

C. Decomposing

In this step, we decompose the algebraic factored form such that each node in the decomposed network has at most three input variables. Any three-variable function can be implemented by at most four majority gates in two levels [11]. Suppose, the

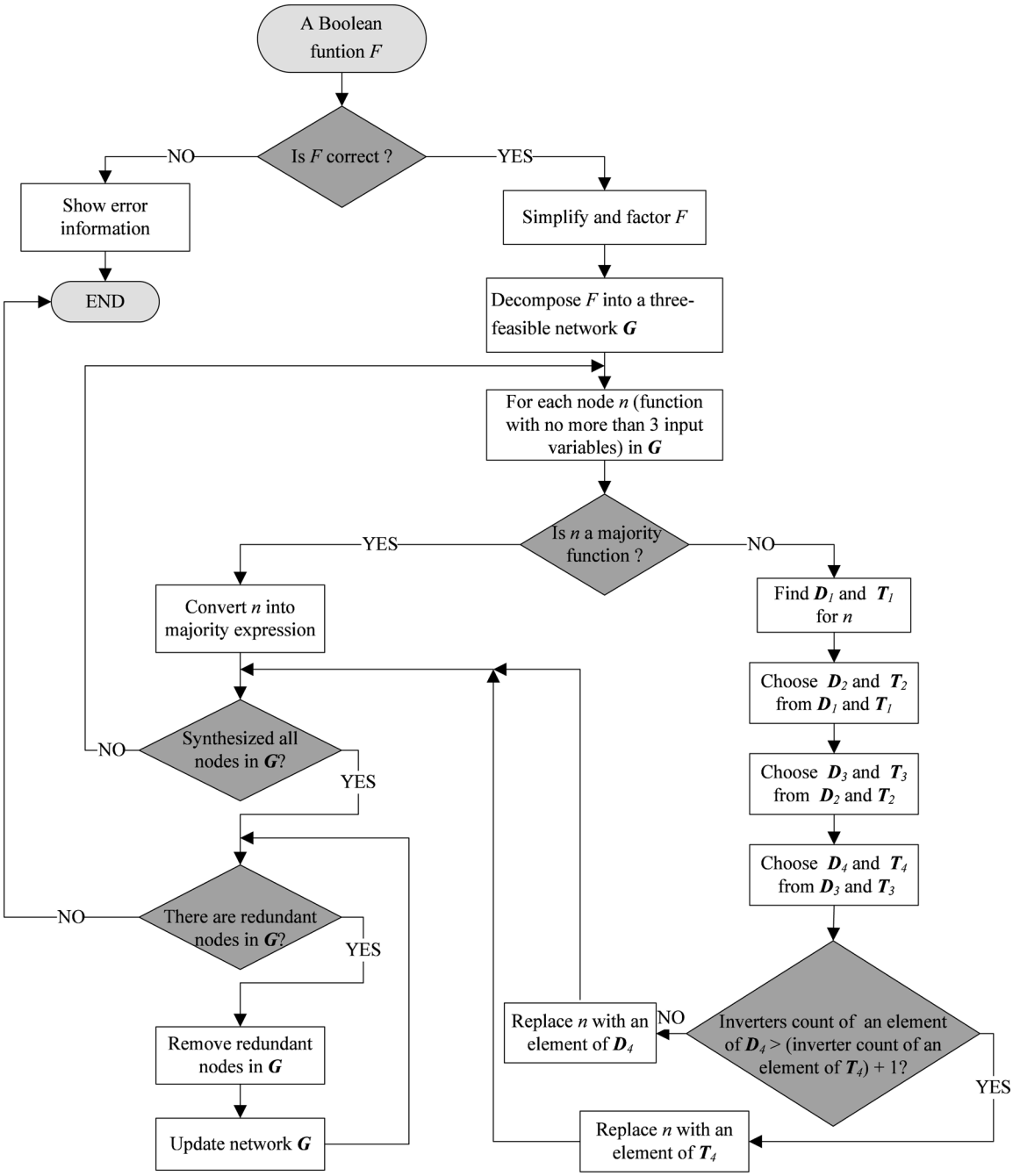

Fig. 5. Flowchart of our majority logic synthesis methodology.

total number of nodes in the decomposed network is N. The total number of majority gates in the synthesized majority network is between N and $4 \times N$. Therefore, in order to reduce the number of majority gates, we should make an effort to reduce N.

In [16] and [17], the SIS synthesis system was used to carry out decomposition. This system is an interactive tool for synthesis and optimization of sequential circuits which integrates many different programs and algorithms [22]. Zhang *et al.* [16] and Huo *et al.* [17] employed the commands "$xl\_split - n3$"(method 1 in Fig. 7) and "$xl\_imp - n3$"(method 2 in Fig. 7) of this SIS system, respectively, to decompose all networks. "$xl\_split - n3$" extracts kernels from a three-infeasible node. This procedure is recursively applied on the kernel and the node until either they become three-feasible or no more kernels can be extracted, in which case a simple AND–OR decomposition is performed. "$xl\_imp - n3$" applies a set of decomposition techniques on each three-infeasible node of the network. The best decomposition result (i.e., the one which has a minimum number of three-feasible nodes) is selected.

After decomposition, a three-feasible network is obtained. In this network, it may be possible to collapse some nodes into their fanouts while retaining feasibility, which can reduce further the node count. However, [16] and [17] do not consider this process that can be achieved by some commands in SIS (such as

1. $collapse$
2. $sweep$
3. $eliminate\ 5$
4. $simplify\ -m\ nocomp\ -d$
5. $resub\ -a\ -d$
6. $gkx\ abt\ 30$
7. $resub\ -a\ -d$
8. $sweep$
9. $gcx\ -bt\ 30$
10. $resub\ -a\ -d$
11. $sweep$
12. $gkx\ -abt\ 10$
13. $resub\ -a\ -d$
14. $sweep$
15. $gcx\ -bt\ 10$
16. $resub\ -a\ -d$
17. $sweep$
18. $gkx\ -ab$
19. $resub\ -a\ -d$
20. $sweep$
21. $gcx\ -b$
22. $resub\ -a\ -d$
23. $sweep$
24. $eliminate\ 0$

Fig. 6. Script for preprocessing Boolean functions.

Method 1: $xl\_split\ -n\ 3$
Method 2: $xl\_imp\ -n\ 3$
Method 3: $xl\_part\_coll\ -n\ 3\ -m\ -g\ 2$
$xl\_coll\_ck\ -n\ 3$
$xl\_partition\ -n\ 3\ -m$
$full\_simplify$
$xl\_imp\ -n\ 3$
$xl\_partition\ -n\ 3\ -t$
$xl\_cover\ -n\ 3\ -e\ 30\ -u\ 200$
$xl\_coll\_ck\ -n\ 3\ -k$
Method 4: $xl\_part\_coll\ -n\ 3\ -m\ -g\ 2$
$xl\_coll\_ck\ -n\ 3$
$xl\_partition\ -n\ 3\ -m$
$sweep;\ eliminate\ -1$
$simplify\ -m\ nocomp$
$eliminate\ -1$
$sweep;\ eliminate\ 5$
$simplify\ -m\ nocomp$
$resub\ -a$
fx
$resub\ -a;\ sweep$
$eliminate\ -1;\ sweep$
$full\_simplify\ -m\ nocomp$
$xl\_imp\ -n\ 3$
$xl\_partition\ -n\ 3\ -t$
$xl\_cover\ -n\ 3\ -e\ 30\ -u\ 200$
$xl\_coll\_ck\ -n\ 3\ -k$

Fig. 7. Four methods for decomposition of Boolean networks.

$xl\_cover, xl\_partition$, etc. [22]). We combine "$xl\_imp\ -n\ 3$" and these commands to form two scripts as our decomposition methods (method 3 and method 4 in Fig. 7).

As we stated above, four methods can be used to process Boolean networks. Therefore, in the next step we should choose the best one to decompose all of the networks. However, in our practical experiments, we find that there is not the best method to decompose all networks efficiently. A simple example is presented to demonstrate this view. Consider the $cm82a$ benchmark from the Microelectronics Center of North Carolina (MCNC) benchmark suite [23]. if we use the four methods in Fig. 7, we get four decomposed networks with 7, 7, 4, and 4 nodes, respectively. Therefore, the third and fourth methods are more proper for the $cm82a$ benchmark. In a similar way, we can obtain four decomposed network with 15, 22, 15, and 16 nodes, respectively, for the $cm85a$ benchmark. The first and third methods are more suitable for it.

Thus, we adopt a new decomposition scheme to deal with the non-universality of decomposition methods. We first use these four different methods, respectively, to decompose a given Boolean network and get four different N. Then, we compare these four numbers and choose the method corresponding to the smallest one as our ultimate decomposition method for the given network. Experiments with 40 MCNC benchmarks show that the average reduction in node count is 15.67%, compared to the existing method [16].

D. Converting

1) Previous Work and Motivational Example: After decomposing the network, our majority synthesis is performed on each node to convert the decomposed network into a majority network. This method is designed for three-feasible functions. As we mentioned earlier, some three-variable majority logic synthesis methods for QCA circuits have been proposed in [12]–[17]. Zhang *et al.* [12] and Walus *et al.* [13] have proposed 13 standard functions to represent all three-variable Boolean functions and presented the majority expressions corresponding to these standard functions. In [14], a method based on genetic algorithm has been employed to reduce the hardware requirements for a QCA design. A technique using the 'disjointing concept' has been presented in [15]. However, these methods are manual. Zhang *et al.* [16] and Huo *et al.* [17] have proposed two automatic QCA majority logic synthesis tools. In [16], the authors employ a K-map-based method to find three one-level majority functions f_1, f_2, and f_3 for each node, such that the function of the node can be represented as $M(f_1, f_2, f_3)$. In [17], a one-to-one mapping table consisting of twenty standard functions and their corresponding majority expressions is employed. For a given three-feasible Boolean function, the authors first simplify the function to one form of these twenty standard functions. Then the majority expression corresponding to this standard function is chosen as the result. A common drawback of these methods is that only one of all admissible majority expressions for a Boolean function can be obtained. Therefore, they can not guarantee that their results are minimal majority expressions. minimal majority expressions are vital for QCA circuits because they can bring cheapest QCA implementations.

We give a simple example to demonstrate the need for a minimal majority expression. Consider the function $f = abc + ab'c'$. If we use the method in [16], then the following formula for f is obtained:

$$f = M(M(M(b', 0, c'), 1, M(b, 0, c)), 0, a).$$

The QCA layout corresponding to this majority formula is shown in Fig. 8(a). It requires seven-clock phases ($1\frac{3}{4}$ clock periods), 146 QCA cells, and an area of $20d \times 23d$ (where d is the center-to-center cell spacing). If we use the method in [17], then the following formula for f is obtained:

$$f = M(M(a, b, c'), 0, M(a, b', c)).$$

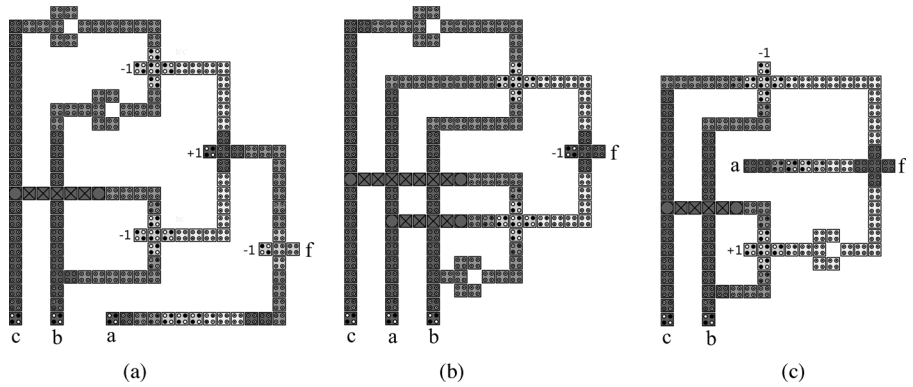

Fig. 8. Three QCA layouts of the function $f = abc + ab'c'$. (a) QCA layout corresponding to the majority formula in [16]. (b) QCA layout corresponding to the majority formula in [17]. (c) QCA layout corresponding to our majority formula.

The QCA layout corresponding to this majority formula is shown in Fig. 8(b). It requires five-clock phases ($1\frac{1}{4}$ clock periods), 154 QCA cells, and an area of $19d \times 23d$.

Furthermore, We find f can be also represented as

$$f = M(M(b, 0, c), a, M(b, 1, c)').$$

The QCA layout corresponding to this majority formula is shown in Fig. 8(c). It requires five-clock phases ($1\frac{1}{4}$ clock periods), 108 QCA cells, and an area of $17d \times 19d$. All the layouts in Fig. 8 were implemented by using QCADesigner [24] and we have simulated them for functional correctness.

Through the discussion above, it can be seen that, compared to the majority formula in [16], the clock period count, cell count, and area are reduced by 28.6%, 26.0%, and 29.8%, respectively, by using the third majority formula. Compared to the majority formula in [17], the cell count reduction and the area reduction are 29.9% and 26.1%, respectively. Therefore, the third majority formula is the minimal majority formula of the three majority formulae above. In the following subsections, we will show that this majority formula can be obtained by our method and prove that it is also a minimal majority formula of all admissible majority formulae for f.

2) Cost of a Majority Expression: A minimal majority expression is a minimum cost majority expression. Therefore, in order to obtain a minimal majority expression, we have to discuss the cost of a majority expression.

In traditional CMOS design, the most widely used model for a multi-level circuit is a Boolean network whose nodes represent functions. These functions are represented as SOP or POS expressions. Thus, the cost of a SOP (POS) expression plays an significant role in implementing CMOS circuits. In QCA design, we use a decomposed network to model a QCA circuit. Each node in the decomposed network is a three-feasible function. These functions are converted into majority expressions. Therefore, the cost of a three-feasible majority expression plays an important role in implementing QCA circuits. In order to get an optimal QCA circuit, we have to make an effort to get its majority expression with the minimum cost. However, no work has been done to discuss this problem.

The cost of a majority expression is the cost of its corresponding QCA circuit. We propose four criteria to measure the cost of a QCA circuit.

1) Unlike an SOP (POS) expression which has two levels, a three-feasible majority expression may have multiple levels. The delay and area of a QCA circuit increase with the increase of its levels. Therefore, we take the number of levels as the primary criterion.

2) In a QCA circuit, a majority gate comprises five QCA cells and corresponds to three majority gate inputs. Thus, the number of majority gates is considered as the secondary criterion.

3) A majority gate input represents a QCA wire, while an inverter is realized on a QCA wire. Thus, we consider majority gate input count and inverter count as the third and fourth criteria, respectively.

We give a simple example to demonstrate the cost of a majority expression. Consider

$$M(M(a, b, c'), M(a, b', c), 1).$$

If we use it to implement a QCA circuit, then two levels, three majority gates, eight gate inputs, and two inverters are needed. Since logic "0" ("1") corresponds to a single cell, it does not correspond to a QCA wire. Therefore, logic "0" ("1") is not considered as a gate input in our criteria.

3) Description of Our Method: Based on the above criteria, we propose our majority synthesis algorithm which is designed for three-feasible functions. The main difference between previous methods and our method is that we can get many admissible majority expressions for a given Boolean function. Therefore, we can compare these expressions and choose the minimal one. Furthermore, a more novel contribution is that we can prove that the minimal one of these majority expressions is also the minimal one of all admissible majority expressions. That is, we can obtain a minimal majority expression for the given function. The main steps of our procedure are as follows.

1) We take all 40 functions that can be realized by a majority gate as the primitives (denoted by $\mathbb{S}$) and then convert them into the minimal majority expressions (denoted by $\mathbb{S}_1$). They are shown in Table I.
2) We denote the logic function of node n by f. If f belongs to $\mathbb{S}$, we convert it into the corresponding majority ex-

TABLE I
FORTY PRIMITIVE FUNCTIONS AND THEIR CORRESPONDING MINIMAL MAJORITY EXPRESSIONS

Function No.	**S:** Primitive Functions	**S_1:** Majority expressions	Function No.	**S:** Primitive Functions	**S_1:** Majority expressions
1	0	0	21	a'b'	M(a, 1, b)'
2	1	1	22	a + b	M(a, 1, b)
3	a	a	23	a'c'	M(a, 1, c)'
4	a'	a'	24	a + c	M(a, 1, c)
5	b	b	25	b'c'	M(b, 1, c)'
6	b'	b'	26	b + c	M(b, 1, c)
7	c	c	27	a'b	M(a', 0, b)
8	c'	c'	28	a + b'	M(a, 1, b')
9	ab	M(a, 0, b)	29	a'c	M(a', 0, c)
10	a' + b'	M(a, 0, b)'	30	a + c'	M(a, 1, c')
11	ac	M(a, 0, c)	31	b'c	M(b', 0, c)
12	a' + c'	M(a, 0, c)'	32	b + c'	M(b, 1, c')
13	bc	M(b, 0, c)	33	ab + ac + bc	M(a, b, c)
14	b' + c'	M(b, 0, c)'	34	a'b' + a'c' +b'c'	M(a, b, c)'
15	ab'	M(a, 0, b')	35	a'b + a'c + bc	M(a', b, c)
16	a' + b	M(a', 1, b)	36	ab' + ac' + b'c'	M(a, b', c')
17	ac'	M(a, 0, c')	37	ab' + ac + b'c	M(a, b', c)
18	a' + c	M(a', 1, c)	38	a'b + a'c' + bc'	M(a', b, c')
19	bc'	M(b, 0, c')	39	ab + ac' + bc'	M(a, b, c')
20	b' + c	M(b', 1, c)	40	a'b' + a'c + b'c	M(a', b', c)

Input: function f with no more than 3 input variables
Output: majority expressions corresponding to f
begin
1 **if** $f \in \mathbb{S}$ **then**
2 convert f into the corresponding majority expression in $\mathbb{S}_1$
3 quit // end the procedure
4 **else**
5 $p \leftarrow 0$
6 **for** $i = 1$ to 36 **do**
7 **for** $j = g(i)$ to 38 **do** // if i is even, then $g(i) = i + 1$. Otherwise, $g(i) = i + 2$.
8 **for** $m = g(j)$ to 40 **do**
9 get the K-map of f and denote it by K_f
10 get the K-maps of the ith, jth, and mth functions in $\mathbb{S}$ and denote them by K_i, K_j, and K_m, respectively
11 **for** $n = 0$ to 7 **do**
12 **if** the nth position of K_f is "1" and the nth positions of K_i, K_j, K_m have at most one "1" **then**
13 $p \leftarrow 0$
14 break
15 **else if** the nth position of K_f is "0" and the nth positions of K_i, K_j, K_m have at least two "1" **then**
16 $p \leftarrow 0$
17 break
18 **else**
19 $p \leftarrow p + 1$
20 **if** $p == 8$ **then**
21 print the majority expression composed of the ith, jth, and mth expressions in $\mathbb{S}_1$
end

Fig. 9. Pseudocode for step 2 of the algorithm.

pression in $\mathbb{S}_1$ and then end the procedure. Otherwise, we find all admissible expression groups from $\mathbb{S}_1$. Each group is composed of three majority expressions (M_1, M_2, M_3) such that f can be represented as their majority function, i.e., $f = M(M_1, M_2, M_3)$. The pseudocode of this process is shown in Fig. 9.

3) We use $\mathbb{D}_1$ to denote the set of majority expressions that are composed of expression groups obtained in the previous step. All the expressions with a minimum number of majority gates are chosen from $\mathbb{D}_1$. We denote the set of these expressions by $\mathbb{D}_2$.
4) All the expressions with a minimum number of gate inputs are chosen from $\mathbb{D}_2$, $\mathbb{D}_3$ is used to denote the set of them.
5) We choose all the expressions with a minimum number of inverters from $\mathbb{D}_3$ and use $\mathbb{D}_4$ to denote the set of them.
6) f' is used to denote the complement of f . We repeat steps 2–5 for f' and then obtain four sets which are denoted by $\mathbb{T}_1, \mathbb{T}_2, \mathbb{T}_3$, and $\mathbb{T}_4$, respectively.
7) We denote the inverter count of a majority expression in $\mathbb{D}_4$ and the inverter count of a majority expression in $\mathbb{T}_4$ by n_1 and n_2, respectively. If $n_1 > n_2 + 1$, then we use the complement of an expression in $\mathbb{T}_4$ to represent f, i.e., $f = M(M_1, M_2, M_3)'$. Otherwise, we use an expression in $\mathbb{D}_4$ to represent f, i.e., $f = M(M_1, M_2, M_3)$.

Since there exists a bijective mapping from $\mathbb{D}_4$ to $\mathbb{T}_4$, we can also obtain $\mathbb{T}_4$ by $\mathbb{D}_4$ directly.

4) Example and Comparison: We present a simple example to illustrate the need for our majority synthesis algorithm. Consider the function $f = ab + a'b'c$. First, we find the set $\mathbb{D}_1$ for f

$$\begin{aligned}\mathbb{D}_1 = \{&M(M(a,0,b), M(a',1,b), M(b',0,c)),\\ &M(M(a,0,b), M(a',1,b), M(a,b',c)),\\ &M(M(a,0,b), M(a,1,b)', M(a,1,c)),\\ &M(M(a,0,b), M(a,1,b)', M(b,1,c)),\\ &M(M(a,0,b), M(a,1,b'), M(a',0,c)),\\ &M(M(a,0,b), M(a,1,b'), M(a',b,c)),\}.\end{aligned}$$

Since each expression in $\mathbb{D}_1$ has four majority gates, $\mathbb{D}_2 = \mathbb{D}_1$. Next, we get $\mathbb{D}_3$ and $\mathbb{D}_4$

$$\begin{aligned}\mathbb{D}_3 = \{&M(M(a,0,b), M(a',1,b), M(b',0,c)),\\ &M(M(a,0,b), M(a,1,b)', M(a,1,c)),\\ &M(M(a,0,b), M(a,1,b)', M(b,1,c)),\\ &M(M(a,0,b), M(a,1,b'), M(a',0,c)),\}\end{aligned}$$

$$\begin{aligned}\mathbb{D}_4 = \{&M(M(a,0,b), M(a,1,b)', M(a,1,c)),\\ &M(M(a,0,b), M(a,1,b)', M(b,1,c))\}.\end{aligned}$$

Then, we obtain $\mathbb{T}_4$

$$\begin{aligned}\mathbb{T}_4 = \{&M(M(a,0,b)', M(a,1,b), M(a,1,c)'),\\ &M(M(a,0,b)', M(a,1,b), M(b,1,c)')\}.\end{aligned}$$

Finally, since each expression in $\mathbb{D}_4$ has one inverter and each expression in $\mathbb{T}_4$ has two inverters, an expression in $\mathbb{D}_4$ is chosen as our last result. Here, we use the first one, i.e., $f = M(M(a,0,b), M(a,1,b)', M(a,1,c))$. This requires two levels, four majority gates, nine gate inputs, and one inverter.

TABLE II
COMPARISON FOR $f = ab + a'b'c$

Standard function	Method	Majority expression	Levels	Majority gates	Gate inputs	Inverters	Delay (clocks)	Cells	Area
F = ab + a'b'c	[12] [15] [16]	M(M(M(a',0,b'),0,c),1, M(a,0,b))	3	4	8	2	$1\frac{3}{4}$	150	$23d \times 26d$
	[13]	M(M(a',1,b),M(a,0,b), M(b',0,c))	2	4	9	2	$1\frac{1}{4}$	155	$17d \times 30d$
	[14]	M(M(a,0,b),M(a,1,b)', M(a,1,c))	2	4	9	1	$1\frac{1}{4}$	131	$16d \times 28d$
	[17]	M(M(a',1,b),M(a,b',c), M(a,0,b))	2	4	10	2	$1\frac{1}{4}$	180	$18d \times 31d$
	Our method	M(M(a,0,b),M(a,1,b)', M(a,1,c))	2	4	9	1	$1\frac{1}{4}$	131	$16d \times 28d$

TABLE III
MORE COMPARISONS FOR SOME STANDARD FUNCTIONS

Standard function	Method	Majority expression	Levels	Majority gates	Gate inputs	Inverters	Dealy (clocks)	Cells	Area
F = a + bc	[12] [13] [15] [16] [17]	M(M(b,0,c),1,a)	2	2	4	0	1	32	$8d \times 12d$
	[14]	M(M(a,1,c),a,b)	2	2	5	0	1	44	$11d \times 13d$
	Our method	M(M(b,0,c),1,a)	2	2	4	0	1	32	$8d \times 12d$

Standard function	Method	Majority expression	Levels	Majority gates	Gate inputs	Inverters	Dealy (clocks)	Cells	Area
F=abc + ab'c'	[12] [13] [17]	M(M(a,b,c'),0,M(a,b',c))	2	3	8	2	$1\frac{1}{4}$	154	$19d \times 23d$
	[14]	M(M(b,0,c),a,M(b,1,c)')	2	3	7	1	$1\frac{1}{4}$	108	$17d \times 19d$
	[15] [16]	M(M(M(b',0,c'),1, M(b,0,c)),0,a)	3	4	8	2	$1\frac{3}{4}$	146	$20d \times 23d$
	Our method	M(M(b,0,c),a,M(b,1,c)')	2	3	7	1	$1\frac{1}{4}$	108	$17d \times 19d$

Standard function	Method	Majority expression	Levels	Majority gates	Gate inputs	Inverters	Dealy (clocks)	Cells	Area
F = abc + a'b'c'	[12] [15] [16]	M(M(M(a,0,b),0,c),1, M(M(a',0,b'),0,c'))	3	5	10	3	$1\frac{3}{4}$	198	$22d \times 31d$
	[13] [17]	M(M(a',1,b),M(b',0,c'), M(a,0,c))	2	4	9	3	$1\frac{1}{4}$	165	$19d \times 31d$
	[14]	M(M(a,1,b)',M(b,0,c), M(a,1,c'))	2	4	9	2	$1\frac{1}{4}$	149	$19d \times 29d$
	Our method	M(M(a,0,b),M(b,1,c)', M(a',1,c))	2	4	9	2	$1\frac{1}{4}$	149	$19d \times 29d$

Standard function	Method	Majority expression	Levels	Majority gates	Gate inputs	Inverters	Dealy (clocks)	Cells	Area
F = abc + a'bc' +ab'c'	[12] [13]	M(M(a,0,c),M(a,b,c'), M(a',b',c'))	2	4	11	4	$1\frac{1}{4}$	210	$19d \times 33d$
	[14]	M(M(M(a,b,c),1,a)',c, M(a,b,c'))	3	4	11	2	$1\frac{3}{4}$	190	$24d \times 29d$
	[15] [16]	M(M(a,b,c'),M(a',0,c'), M(a,b',c))	2	4	11	4	$1\frac{1}{4}$	211	$19d \times 33d$
	[17]	M(M(a,b,c'),M(a,b',c), M(a',0,b))	2	4	11	3	$1\frac{1}{4}$	200	$19d \times 33d$
	Our method	M(M(a,0,c),M(a,b,c'), M(a,b,c'))	2	4	11	2	$1\frac{1}{4}$	182	$20d \times 29d$

Standard function	Method	Majority expression	Levels	Majority gates	Gate inputs	Inverters	Dealy (clocks)	Cells	Area
F = ab + b'c	[12] [13] [15] [16] [17]	M(M(a,0,b),1,M(b',0,c))	2	3	6	1	$1\frac{1}{4}$	68	$12d \times 18d$
	[14]	M(M(a,1,b)',a,M(b,1,c))	2	3	7	1	$1\frac{1}{4}$	77	$13d \times 19d$
	Our method	M(M(a,0,b),1,M(b',0,c))	2	3	6	1	$1\frac{1}{4}$	68	$12d \times 18d$

Standard function	Method	Majority expression	Levels	Majority gates	Gate inputs	Inverters	Dealy (clocks)	Cells	Area
F=ab + bc + a'b'c'	[12]	M(M(M(a,1,c),0,b),1, M(a',0,M(b'0,c')))	3	5	10	3	$1\frac{3}{4}$	195	$23d \times 30d$
	[13] [17]	M(M(a,b,c),M(a',1,b), M(b',0,c'))	2	4	10	3	$1\frac{1}{4}$	156	$16d \times 30d$
	[15] [16]	M(M(a,b,c),M(a',0,b'), M(b,1,c'))	2	4	10	3	$1\frac{1}{4}$	156	$16d \times 30d$
	Our method	M(M(a',1,b),M(b,1,c)', M(a,b,c))	2	4	10	2	$1\frac{1}{4}$	147	$16d \times 29d$

Standard function	Method	Majority expression	Levels	Majority gates	Gate inputs	Inverters	Dealy (clocks)	Cells	Area
F = abc + a'b'c +ab'c'+a'bc'	[12] [13][17]	M(M(a',b,c),c',M(a,b',c))	2	3	9	3	$1\frac{1}{4}$	158	$18d \times 26d$
	[14]	M(M(a,b,c)',c,M(a,b,c'))	2	3	9	2	$1\frac{1}{4}$	149	$18d \times 23d$
	[15] [16]	M(M(a',b,c),M(a,b,c'), M(a,b',c))	2	4	12	3	$1\frac{1}{4}$	198	$18d \times 30d$
	Our method	M(M(a,b,c)',a,M(a',b,c))	2	3	9	2	$1\frac{1}{4}$	149	$18d \times 23d$

If we use the methods in [12], [15], [16], then the following formula for f is obtained:

$$f = M(M(a, 0, b), 1, M(M(a', 0, b'), 0, c)).$$

It requires three levels, four majority gates, eight gate inputs, and two inverters. If we use the method in [13], then the following result is obtained:

$$f = M(M(b', 0, c), M(a', 1, b), M(a, 0, b)).$$

It requires two levels, four majority gates, nine gate inputs, and two inverters. If we use the method in [17], then the following formula is obtained:

$$f = M(M(a', 1, b), M(a, b', c), M(a, 0, b)).$$

It requires two levels, four majority gates, ten gate inputs, and two inverters.

We have implemented the QCA layouts of the above majority formulae and compared them. The comparison is summarized as Table II. More comparisons with some standard functions [12] are shown in Table III. From these tables, it can be seen that for a given standard function, the cost of the majority expression obtained by our method is not higher than any one of the majority expressions obtained by existing methods. Furthermore, the QCA circuit corresponding to our result is optimal.

5) Proof of Minimal Majority Expression: We have proven that, for any three-feasible function, its corresponding minimal majority expression can be obtained by our method. The detailed process of our proof can be seen in the Appendix.

Input: majority network G
Output: updated majority network G without redundant nodes
begin
1 $i \leftarrow 0$
2 $p \leftarrow 1$
3 **while** $p > 0$ **do**
4 $p \leftarrow 0$
5 **while** $i < |G|-1$ **do**
6 $n_i' \leftarrow$ complement of node n_i
7 **for** $j = i+1$ to $|G|-1$ **do**
8 **if** $n_i = n_j$ **then**
9 replace node n_j with literal n_i
10 $p \leftarrow p+1$
11 **else if** $n_i' = n_j$ **then**
12 replace node n_j with literal n_i'
13 $p \leftarrow p+1$
14 **else**
15 continue
16 **if** $p > 0$ **then**
17 sweep G // remove all the single-input internal nodes of G
18 break
19 **else**
20 $i \leftarrow i+1$
end

Fig. 10. Algorithm for removing redundancies in a majority network.

E. Removing

By synthesizing all nodes in the decomposed network, we convert the network into a majority network. In this majority network, there exist some nodes with the same or complementary functions, which causes a waste of QCA cells in the ultimate QCA circuit.

We give a simple example to demonstrate this problem. Consider the $cm82a$ benchmark of the MCNC benchmark suite. We first preprocess the benchmark circuit and then decompose it with method 4 in Fig. 7. The following decomposed network is obtained:

$$\begin{cases} \{f\} = abc + ab'c' + a'bc' + a'b'c \\ \{g\} = (1)de + (1)d'e' + (1)'de' + (1)'d'e \\ \{h\} = (1)d + (1)e + de \\ (1) = ab + ac + bc \end{cases}$$

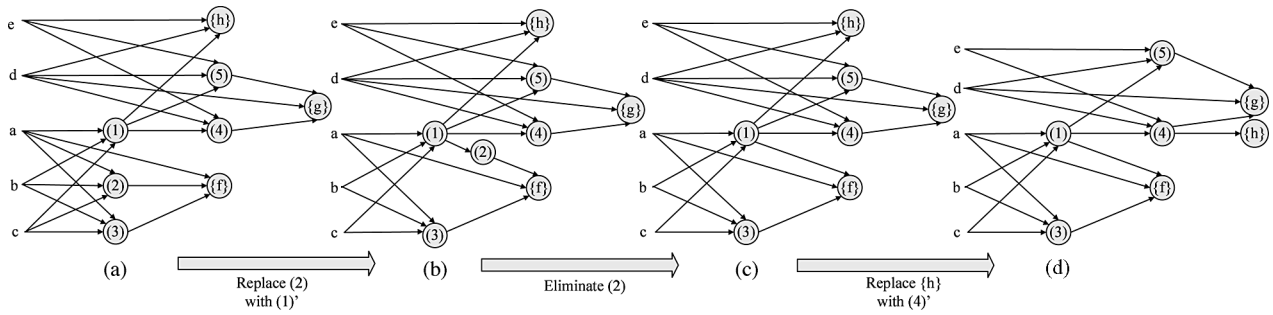

Fig. 11. Process of removing redundancies in the majority network.

TABLE IV
EXPERIMENTAL RESULTS AND COMPARISONS FOR 40 MCNC BENCHMARKS WHERE FOUR METHODS ARE USED

Benchmark	Method	Levels	Majority gates	Gate inputs	Inverters	Time(s)
b1	[16]	3	9	18	6	< 0.1
	(1)	2	7	17	5	< 0.1
	(2)	2	7	17	5	< 0.1
	(3)	2	7	17	5	< 0.1
Reduction (%R)		33.3	22.2	5.6	16.7	
cm42a	[16]	2	21	42	12	< 0.1
	(1)	2	21	42	9	< 0.1
	(2)	2	21	42	9	< 0.1
	(3)	2	18	36	6	< 0.1
Reduction (%R)		0	14.3	14.3	50.0	
decod	[16]	3	28	56	8	< 0.1
	(1)	3	28	56	6	< 0.1
	(2)	3	28	56	6	< 0.1
	(3)	3	28	56	6	< 0.1
Reduction (%R)		0	0	0	25.0	
cm82a	[16]	4	16	39	10	< 0.1
	(1)	4	14	35	6	< 0.1
	(2)	3	8	24	4	< 0.1
	(3)	3	7	19	6	< 0.1
Reduction (%R)		25.0	56.3	51.3	40.0	
majority	[16]	4	6	13	0	< 0.1
	(1)	4	6	13	0	< 0.1
	(2)	4	6	13	0	< 0.1
	(3)	4	6	13	0	< 0.1
Reduction (%R)		0	0	0	0	
z4ml	[16]	4	27	63	16	< 0.1
	(1)	4	24	58	11	< 0.1
	(2)	4	12	36	6	0.1
	(3)	4	9	27	6	0.1
Reduction (%R)		0	66.7	57.1	62.5	
9symml	[16]	12	216	428	87	0.2
	(1)	12	214	428	79	0.2
	(2)	10	51	116	25	0.9
	(3)	10	47	105	23	0.9
Reduction (%R)		16.7	78.2	75.5	73.6	
ldd	[16]	7	73	146	27	0.1
	(1)	7	73	146	27	0.1
	(2)	7	73	146	27	0.1
	(3)	7	67	134	24	0.1
Reduction (%R)		0	8.2	8.2	11.1	
alu2	[16]	18	356	712	132	0.3
	(1)	18	356	712	110	0.4
	(2)	18	356	712	110	0.4
	(3)	18	340	680	107	1.3
Reduction (%R)		0	4.5	4.5	18.9	
x2	[16]	7	42	84	24	< 0.1
	(1)	7	41	83	17	< 0.1
	(2)	7	41	83	17	< 0.1
	(3)	7	37	75	15	< 0.1
Reduction (%R)		0	11.9	10.7	37.5	
cm152a	[16]	6	21	42	7	< 0.1
	(1)	6	21	42	7	< 0.1
	(2)	6	21	42	7	< 0.1
	(3)	6	21	42	7	< 0.1
Reduction (%R)		0	0	0	0	
cm85a	[16]	7	34	72	18	< 0.1
	(1)	7	34	72	14	< 0.1
	(2)	6	26	60	12	< 0.1
	(3)	6	26	60	12	< 0.1
Reduction (%R)		14.3	23.5	16.7	33.3	
cm151a	[16]	7	42	84	18	< 0.1
	(1)	7	42	84	17	< 0.1
	(2)	7	23	45	10	0.1
	(3)	7	23	45	10	0.1
Reduction (%R)		0	45.2	46.4	44.4	
cm162a	[16]	7	46	92	21	< 0.1
	(1)	7	46	92	13	< 0.1
	(2)	7	46	92	13	0.1
	(3)	7	41	82	14	0.1
Reduction (%R)		0	10.9	10.9	33.3	
cu	[16]	7	46	92	28	< 0.1
	(1)	7	46	92	22	< 0.1
	(2)	7	42	84	22	0.1
	(3)	7	40	80	21	0.1
Reduction (%R)		0	13.0	13.0	25.0	
cm163a	[16]	7	42	84	22	< 0.1
	(1)	7	42	84	15	< 0.1
	(2)	7	42	84	15	< 0.1
	(3)	7	38	76	17	< 0.1
Reduction (%R)		0	9.5	9.5	22.7	
cmb	[16]	4	44	88	24	< 0.1
	(1)	4	44	88	12	< 0.1
	(2)	4	44	88	12	< 0.1
	(3)	4	28	55	4	0.1
Reduction (%R)		0	36.4	37.5	83.3	
pm1	[16]	6	45	87	24	< 0.1
	(1)	6	45	87	18	< 0.1
	(2)	6	38	74	17	< 0.1
	(3)	6	35	68	16	0.1
Reduction (%R)		0	22.2	21.8	33.3	
tcon	[16]	2	24	48	8	< 0.1
	(1)	2	24	48	8	< 0.1
	(2)	2	24	48	8	< 0.1
	(3)	2	24	48	8	< 0.1
Reduction (%R)		0	0	0	0	
vda	[16]	16	739	1478	147	1.7
	(1)	16	739	1478	131	1.7
	(2)	15	727	1454	151	2.1
	(3)	15	700	1400	142	6.4
Reduction (%R)		6.3	5.3	5.3	3.4	
pcle	[16]	8	67	134	22	< 0.1
	(1)	8	67	134	18	< 0.1
	(2)	8	63	129	17	0.2
	(3)	8	62	127	17	0.2
Reduction (%R)		0	7.5	5.2	22.7	
sct	[16]	6	72	146	28	< 0.1
	(1)	6	72	147	23	< 0.1
	(2)	6	72	147	23	< 0.1
	(3)	6	65	133	21	0.1
Reduction (%R)		0	9.7	8.9	25.0	
cc	[16]	5	44	92	10	< 0.1
	(1)	5	44	92	8	< 0.1
	(2)	5	44	92	8	< 0.1
	(3)	5	43	89	8	< 0.1
Reduction (%R)		0	2.3	3.3	20.0	
cm150a	[16]	9	46	92	29	< 0.1
	(1)	9	46	92	20	< 0.1
	(2)	9	46	92	20	< 0.1
	(3)	9	46	92	20	< 0.1
Reduction (%R)		0	0	0	31.0	
mux	[16]	9	46	92	13	< 0.1
	(1)	9	46	92	12	< 0.1
	(2)	9	46	92	12	< 0.1
	(3)	9	46	92	12	< 0.1
Reduction (%R)		0	0	0	7.7	
ttt2	[16]	11	154	308	69	0.1
	(1)	11	152	307	54	0.1
	(2)	11	152	307	54	0.1
	(3)	11	145	293	52	0.2
Reduction (%R)		0	5.8	4.9	24.6	
i1	[16]	6	41	82	13	< 0.1
	(1)	6	41	82	10	< 0.1
	(2)	6	40	81	10	0.1
	(3)	6	36	73	12	0.1
Reduction (%R)		0	12.2	11.0	7.7	
lal	[16]	8	95	191	44	0.1
	(1)	8	95	192	35	0.1
	(2)	8	95	192	35	0.1
	(3)	8	82	165	32	0.2
Reduction (%R)		0	13.7	13.6	27.3	
pcler8	[16]	10	90	180	20	< 0.1
	(1)	10	90	180	17	0.1
	(2)	9	88	179	18	0.3
	(3)	9	80	163	15	0.4
Reduction (%R)		10.0	11.1	9.4	25.0	
frg1	[16]	18	111	222	95	0.1
	(1)	18	111	222	64	0.2
	(2)	18	111	222	64	0.2
	(3)	18	105	210	61	0.5
Reduction (%R)		0	5.4	5.4	35.8	
c8	[16]	7	115	230	44	0.1
	(1)	7	115	230	30	0.1
	(2)	7	115	230	30	0.1
	(3)	7	112	224	28	0.1
Reduction (%R)		0	2.6	2.6	36.4	
term1	[16]	13	174	347	85	0.2
	(1)	13	173	347	64	0.3
	(2)	11	112	235	54	0.3
	(3)	11	106	223	50	0.7
Reduction (%R)		15.4	39.1	35.7	41.2	
unreg	[16]	5	84	168	50	< 0.1
	(1)	5	84	168	33	< 0.1
	(2)	5	84	168	33	< 0.1
	(3)	5	84	168	33	0.1
Reduction (%R)		0	0	0	34.0	
k2	[16]	27	1313	2622	240	18.2
	(1)	27	1311	2618	215	19.1
	(2)	19	1301	2602	219	19.6
	(3)	19	1301	2602	219	19.9
Reduction (%R)		29.6	0.8	0.8	8.8	
cht	[16]	4	120	240	12	0.1
	(1)	4	120	240	9	0.1
	(2)	4	120	240	9	0.1
	(3)	4	120	240	9	0.1
Reduction (%R)		0	0	0	25.0	
x1	[16]	11	320	647	136	0.2
	(1)	11	320	647	96	0.2
	(2)	11	288	583	104	1.6
	(3)	11	264	531	95	2.3
Reduction (%R)		0	17.5	17.9	30.1	
example2	[16]	10	259	518	75	0.2
	(1)	10	259	518	67	0.2
	(2)	10	259	518	67	0.2
	(3)	10	247	494	63	0.6
Reduction (%R)		0	4.6	4.6	16.0	
apex6	[16]	17	701	1402	204	0.8
	(1)	17	700	1401	175	0.8
	(2)	17	692	1388	166	0.9
	(3)	17	662	1328	158	4.1
Reduction (%R)		0	5.6	5.3	22.5	
frg2	[16]	15	672	1338	136	3.3
	(1)	15	671	1337	120	3.5
	(2)	14	589	1176	137	5.1
	(3)	14	582	1162	133	6.4
Reduction (%R)		6.7	13.4	13.2	2.2	
i2	[16]	19	209	418	0	0.2
	(1)	19	209	418	0	0.4
	(2)	13	209	418	0	0.5
	(3)	13	209	418	0	0.6
Reduction (%R)		31.6	0	0	0	

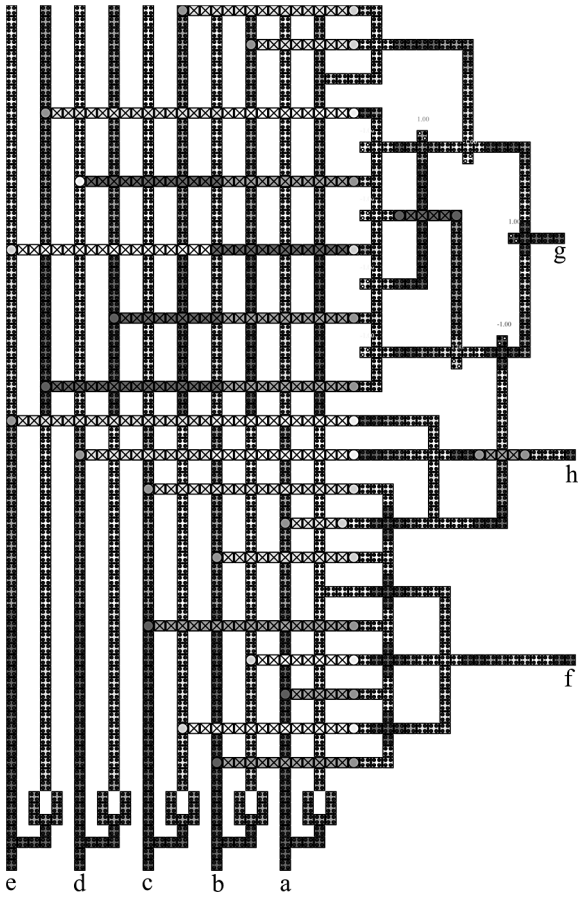

Fig. 12. QCA layout of the $cm82a$ benchmark (implemented by the existing method).

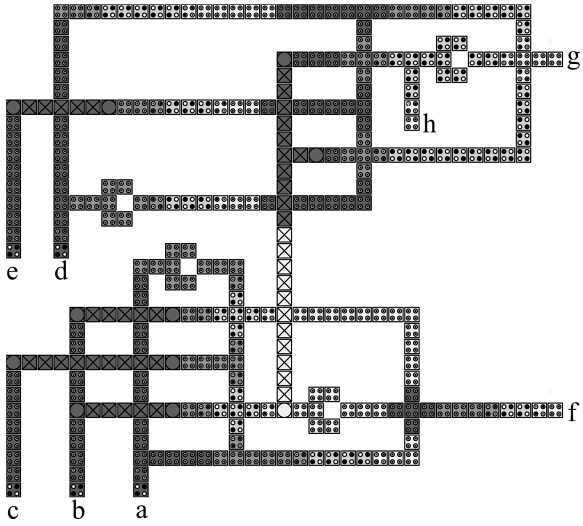

Fig. 13. QCA layout of the $cm82a$ benchmark (implemented by our method).

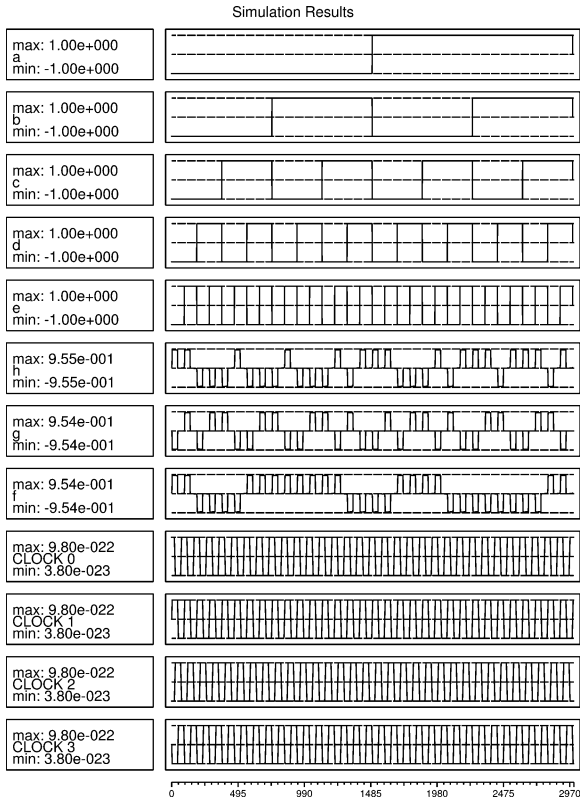

Fig. 14. Simulation waveforms for the $cm82a$ benchmark (simulated by QCADesigner).

where $\{f\}, \{g\}$, and $\{h\}$ are output nodes, (1) is an internal node.

Then, we synthesize this network by using our majority synthesis algorithm. The following network is obtained:

$$\begin{cases} \{f\} = (2)(3) + (2)a + (3)a = M((2),(3),a) \\ \{g\} = (4)(5) + (4)d + (5)d = M((4),(5),d) \\ \{h\} = (1)d + (1)e + de = M((1),d,e) \\ (1) = ab + ac + bc = M(a,b,c) \\ (2) = a'b' + a'c' + b'c' = M(a,b,c)' \\ (3) = a'b + a'c + bc = M(a',b,c) \\ (4) = (1)'d' + (1)'e' + d'e' = M((1),d,e)' \\ (5) = (1)d' + (1)e + d'e = M((1),d',e) \end{cases}$$

where $\{f\}, \{g\}$, and $\{h\}$ are output nodes, $(1), (2), (3), (4)$, and (5) are internal nodes.

In the above majority network, (2) and (4) are the complements of (1) and $\{h\}$, respectively. If we design the QCA circuit according to this majority network, we have to design two QCA majority gates for $M(a,b,c)$ and two QCA majority gates for $M((1),d,e)$. It is clear that two QCA majority gates are redundant. However, to our best knowledge, no work has been done to consider these redundancies. Furthermore, no commands in SIS can be used to remove these redundant majority gates and keep the majority network. Thus, we propose an algorithm to remove these redundancies. The main idea of this algorithm is as follows. For any two nodes n_i and n_j $(i < j)$, if the logic functions of them are the same, then we replace n_j with n_i and eliminate n_j from the network. Else if their logic functions are complementary, then we replace n_j with n_i' and eliminate n_j from the network. The pseudocode for this procedure is shown in Fig. 10.

TABLE V
COMPARISONS OF QCA LAYOUTS FOR 10 MCNC BENCHMARKS

Benchmark	Previous method [16]			Our method			Reduction		
	Delay (clock periods)	Cells	Area	Delay (clock periods)	Cells	Area	%R1	%R2	%R3
b1	$1\frac{3}{4}$	282	$20d \times 51d$	$1\frac{1}{4}$	211	$18d \times 38d$	28.6	25.2	32.9
cm42a	$1\frac{1}{2}$	1320	$36d \times 113d$	$1\frac{1}{2}$	1132	$42d \times 90d$	0	14.2	7.1
decod	2	1382	$56d \times 84d$	2	1372	$56d \times 84d$	0	0.7	0
cm82a	3	1597	$66d \times 80d$	2	372	$35d \times 37d$	33.3	76.7	75.5
majority	$2\frac{1}{4}$	201	$25d \times 33d$	$2\frac{1}{4}$	201	$25d \times 33d$	0	0	0
z4ml	3	2818	$65d \times 143d$	3	807	$49d \times 54d$	0	71.4	71.5
cm152a	$3\frac{1}{2}$	756	$36d \times 93d$	$3\frac{1}{2}$	756	$36d \times 93d$	0	0	0
cm85a	4	2205	$109d \times 87d$	$3\frac{1}{2}$	1664	$90d \times 69d$	12.5	24.5	34.5
cm151a	4	3045	$106d \times 113d$	$3\frac{3}{4}$	721	$41d \times 91d$	6.3	76.3	68.9
tcon	$1\frac{1}{2}$	382	$19d \times 63d$	$1\frac{1}{2}$	382	$19d \times 63d$	0	0	0

We use our algorithm on the above majority network. The final simplified network is obtained.

$$\begin{cases} \{f\} = (1)'(3) + (1)'a + (3)a = M((1)', (3), a) \\ \{g\} = (4)(5) + (4)d + (5)d = M((4), (5), d) \\ \{h\} = (4)' \\ (1) = ab + ac + bc = M(a, b, c) \\ (3) = a'b + a'c + bc = M(a', b, c) \\ (4) = (1)'d' + (1)'e' + d'e' = M((1), d, e)' \\ (5) = (1)d' + (1)e + d'e = M((1), d', e) \end{cases}$$

where $\{f\}, \{g\}$, and $\{h\}$ are output nodes, $(1), (3), (4)$, and (5) are internal nodes. Fig. 11 illustrates this process in detail. The original majority network structure is shown in Fig. 11(a). First, we replace node (2) by $(1)'$, as shown in Fig. 11(b). Then, we eliminate node (2) from this network, as shown in Fig. 11(c). Finally, node $\{h\}$ is replaced by $(4)'$, as shown in Fig. 11(d). Since $\{h\}$ is an output node, we need not eliminate it from the network.

From Fig. 11 it can be seen that our method reduces two majority gates and five gate inputs for the $cm82a$ benchmark. Our experiments with more benchmarks [23] indicate that, by applying this method, the average reductions in gate count, gate input count, and inverter count are 5.8%, 6.26%, and 5.66%, respectively.

IV. EXPERIMENTAL RESULTS AND COMPARISONS

We have integrated our methodology within SIS. The package contains about 1100 lines of C code.

A. Experiments and Comparisons on MCNC Benchmarks

We used 40 MCNC benchmarks [23] to test our methodology. These experiments were done on a 2.4-GHz Intel Pentium IV machine with 2048-MB RAM running Ubuntu 8.04. Table IV lists the results for these benchmarks. In this table, [16] represents the existing method in [16], (1) refers to our method with the same decomposition method as [16]. (2) refers to our method with a more suitable decomposition method. (3) refers to method (2) with redundancy removal. *%R* represent the level count reduction, the majority gate count reduction, the gate input count reduction, and the inverter count reduction in the majority network obtained by our majority synthesis method (method (3) in Table IV) compared to the majority network obtained by the existing method [16], respectively.

B. Comparisons of QCA Layouts

In order to visualize the efficiency of our methodology, two QCA layouts for the $cm82a$ benchmark are shown in Figs. 12 and 13. They correspond to the existing method and our method, respectively. We implemented these two layouts by using QCADesigner which provides a design and simulation environment for the QCA circuits [24], [25]. QCADesigner has two types of simulation engines: coherence vector and bistable approximation. In this subsection, we use the coherence vector engine to simulate the two layouts and Fig. 14 shows the simulation results. Input (a) is given by 00000000000000001111111111111111, Input (b) is given by 00000000111111110000000011111111, Input (c) is given by 00001111000011110000111100001111, Input (d) is given by 00110011001100110011001100110011, Input (e) is given by 01010101010101010101010101010101, Output (f–h) are XX00001111111100001111000000011, XX011001100110 100101101001100110, and XX00010001000101110001011 1011101, respectively, where X indicates a do not care condition. This is just the logic behavior of the cm82a benchmark. There is a delay of two-clock periods because it takes two-clock periods for the input to reach the output.

More comparisons of QCA layouts for some benchmarks are shown in Table V. In this table, delay and cells refer to the number of clock periods and cells, area refers to the area of a QCA layout. *%R*1, *%R*2, and *%R*3 represent the delay reduction, the cell count reduction, and the area reduction of the

QCA layout corresponding to our method compared to the QCA layout corresponding to the existing method [16], respectively.

From Table V, we can see that, compared to the existing method, up to 33.3%, 76.7%, and 75.5% reduction in delay, cell count, and area, respectively, is possible with the average being 8.1%, 28.9%, and 29.0%, respectively.

V. Conclusion

In this paper, we introduced a new majority network synthesis methodology for majority gate-based QCA circuit design. We have implemented our methodology on top of some existing logic synthesis tools (SIS [22], MALS [16], etc.) to produce an optimized majority network synthesis tool. An algorithm has been proposed to convert a given three-variable Boolean function into its minimum cost majority expression. Furthermore, we have introduced a procedure to remove the redundancies which are brought by majority network synthesis. Experimental results for MCNC benchmarks show that our methodology offers higher quality majority networks and QCA circuits compared to existing methods.

Since majority networks can be easily converted into minority networks by De Morgan's theorem, our methodology can be extended to TPL- and SET- based nanoscale circuit design. It is expected that our method and tool will produce significant hardware savings for many future QCA-, TPL-, and SET-based architectures.

Appendix

Proof of Minimal Majority Expression

The symbols that are used in our proof are defined as follows:

f	three-feasible function;
F	majority expression which is obtained by our algorithm for f;
$\mathbb{F}$	set of all three-feasible functions;
$\mathbb{P}$	set of all admissible two-level majority expressions corresponding to f;
$\mathbb{A}$	set of all admissible majority expressions corresponding to f;
$\mathbb{T}_1'$	set of the complements of all the majority expressions in $\mathbb{T}_1$;
$\mathbb{Q}_1$	union of set $\mathbb{D}_1$ and $\mathbb{T}_1'$;

the definitions of $\mathbb{D}_1, \mathbb{T}_1, \mathbb{S}$, and $\mathbb{S}_1$ are the same as their definitions in the previous sections.

We first prove the following three lemmas.

Lemma 1: For any $f \in \mathbb{F}$, if $f \notin \mathbb{S}$, then F is a minimum cost element of set $\mathbb{P}$.

Proof: Since $f \notin \mathbb{S}$, none of $\mathbb{D}_1, \mathbb{T}_1, \mathbb{T}_1'$, and $\mathbb{Q}_1$ is an empty set. Let E be an element of $\mathbb{P}$. Then $E = M(E_1, E_2, E_3)$ or $E = M(E_1, E_2, E_3)'$, where $E_i, i \in \{1, 2, 3\}$, is a one-level majority expression, literal, or constant. If $E = M(E_1, E_2, E_3)$, since the function expressed by E_i belongs to $\mathbb{S}$, we can convert E_i into the corresponding expression E_i^* in $\mathbb{S}_1$. Then, we get a new majority formula $E^* = M(E_1^*, E_2^*, E_3^*)$ and $E^* \in \mathbb{D}_1$. Since the cost of E_i^* is not higher than the cost of E_i, the cost of E^* is not higher than the cost of E. In the same way we can get $E^* = M(E_1^*, E_2^*, E_3^*)'$ for $E = M(E_1, E_2, E_3)'$. $E^* \in \mathbb{T}_1'$ and the cost of E^* is not higher than the cost of E. Therefore, for each member of $\mathbb{P}$, a corresponding expression with less or equal cost can be found in $\mathbb{Q}_1$. We define the minimum cost of $\mathbb{Q}_1$ ($\mathbb{P}$) as the cost of a minimum cost member of $\mathbb{Q}_1$ ($\mathbb{P}$).

1) Let K be a minimum cost member of $\mathbb{P}$. Let K^* be K's corresponding member in $\mathbb{Q}_1$. Then the cost of K^* is not higher than the cost of K. Thus, the minimum cost of $\mathbb{Q}_1$ is not higher than the minimum cost of $\mathbb{P}$.
2) Since $\mathbb{Q}_1 \neq \emptyset$ and each member of $\mathbb{Q}_1$ has two levels, $\mathbb{Q}_1 \subseteq \mathbb{P}$. Therefore, the minimum cost of $\mathbb{Q}_1$ is not lower than the minimum cost of $\mathbb{P}$.

From (1) and (2) it can be concluded that the minimum cost of $\mathbb{Q}_1$ is equal to the minimum cost of $\mathbb{P}$. According to our algorithm, F is a minimum cost member of $\mathbb{Q}_1$. Thus F is also a minimum cost member of $\mathbb{P}$. ■

Lemma 2: Let H be a three-level majority expression with three majority gates, and H has at most three variables. Then H can be converted into a two-level majority expression with two majority gates.

Proof: Since H is a three-level majority expression with three majority gates and it has at most three variables, suppose that $H = M(M_1, l, m)$, where $l, m \in \{0, 1, a, b, c, a', b', c'\}$, M_1 is a two-level majority expression with two majority gates and it has at most three variables. The function realized by M_1 can only be one of the following forms [17]:

a) $M_1 = con = M(M(con, 0, 1), 0, 1)$
b) $M_1 = x = M(M(x, 0, 1), 0, 1)$
c) $M_1 = xy = M(M(x, 0, y), 0, 1)$
d) $M_1 = x + y = M(M(x, 1, y), 0, 1)$
e) $M_1 = xy + xz + yz = M(M(x, y, z), 0, 1)$
f) $M_1 = xyz = M(M(x, 0, y), 0, z)$
g) $M_1 = x + y + z = M(M(x, 1, y), 1, z)$
h) $M_1 = x + yz = M(M(y, 0, z), 1, x)$
i) $M_1 = xy + xz = M(M(y, 1, z), 0, x)$

where $con \in \{0, 1\}, x, y, z \in \{a, b, c, a', b', c'\}, x \neq y, x \neq y', x \neq z, x \neq z', y \neq z$, and $y \neq z'$.

If $M_1 = con$, then

$$H = M(M(M(con, 0, 1), 0, 1), l, m) = M(M(con, 0, 1), l, m).$$

If $M_1 = x$, then

$$H = M(M(M(x, 0, 1), 0, 1), l, m) = M(M(x, 0, 1), l, m).$$

If $M_1 = xy$, then

$$H = M(M(M(x, 0, y), 0, 1), l, m) = M(M(x, 0, y), l, m).$$

If $M_1 = x + y$, then

$$H = M(M(M(x, 1, y), 0, 1), l, m) = M(M(x, 1, y), l, m).$$

If $M_1 = xy + xz + yz$, then

$$H = M(M(M(x, y, z), 0, 1), l, m)$$
$$= M(M(x, y, z), l, m).$$

If $M_1 = xyz$, then

$$H = M(M(M(x, 0, y), 0, z), l, m)$$
$$= lm + lxyz + mxyz.$$

In this case, if $l = x$, then

$$H = xm + xyz + mxyz$$
$$= xm + xyz$$
$$= x(m + yz)$$

$$x(m + yz) = \begin{cases} M(M(x, 0, y), 0, z), & \text{if } m = 0 \text{ or } x' \\ M(M(x, 0, 1), 0, 1), & \text{if } m = 1 \text{ or } x \\ M(M(x, 0, y), 0, 1), & \text{if } m = y \\ M(M(y', 1, z), 0, x), & \text{if } m = y' \\ M(M(x, 0, z), 0, 1), & \text{if } m = z \\ M(M(y, 1, z'), 0, x), & \text{if } m = z' \end{cases}$$

If $l = 0$, then $H =$

$$mxyz = \begin{cases} M(M(x, 0, y), 0, z), & \text{if } m = 1, x, y, \text{ or } z \\ M(M(0, 0, 1), 0, 1), & \text{if } m = 0, x', y', \text{ or } z' \end{cases}$$

For the other assumptions ($l = x', l = 1, l = y, l = y', l = z$, and $l = z'$), similar arguments show that H can also be converted into a two-level majority expression with two majority gate.

For the other forms ($M_1 = x + y + z, M_1 = x + yz$, and $M_1 = xy + xz$), similar arguments as $M_1 = xyz$ show that the result holds as well. ■

Lemma 3: For any $f \in \mathbb{F}$, if $f \notin \mathbb{S}$, then F is a minimum cost element of set $\mathbb{A}$.

Proof 3: (1) Let F_1 be an n-level ($n \geq 4$) majority expression for f. Since an n-level ($n \in \mathbb{N}$) majority expression has at least n majority gates, F_1 have at least four levels and four majority gates. However, F has two levels and at most four majority gates. Hence the cost of F is lower than the cost of F_1.

(2) Let F_2 be a three-level majority expression for f. If F_2 has at least four majority gate, then the cost of F_2 is higher than the cost of F. Otherwise, F_2 has three majority gate. According to Lemma 2, F_2 can be converted into a two level majority expression with two gates. We denote this expression by F_3, then $F_3 \in \mathbb{P}$. According to Lemma 1, F is a minimum cost member of $\mathbb{P}$. Therefore, the cost of F is not higher than the cost of F_3. Furthermore, the cost of F is lower than the cost of F_2.

According to (1), (2), and Lemma 1, F is a minimum cost element of set $\mathbb{A}$. ■

Then, we prove our conclusion.

Theorem: For any $f \in \mathbb{F}$, F is a minimal majority expression.

Proof: (1) If $f \in \mathbb{S}$, then $F \in \mathbb{S}_1$. $\mathbb{S}_1$ is composed of minimal majority expressions. Thus, F is a minimal majority expression.

(2) If $f \notin \mathbb{S}$, then F is a minimum cost expression in $\mathbb{A}$ by Lemma 3. Since $\mathbb{A}$ is the set of all admissible majority expressions for f, F is a minimal majority expression. ■

REFERENCES

[1] International Technology Roadmap for Semiconductors (ITRS). [Online]. Available: http://www.itrs.net

[2] P. D. Tougaw and C. S. Lent, "Logical devices implemented using quantum cellular automata," *J. Appl. Phys.*, vol. 75, no. 3, pp. 1818–1825, Feb. 1994.

[3] C. S. Lent and P. D. Tougaw, "A device architecture for computing with quantum dots," *Proc. IEEE*, vol. 85, no. 4, pp. 541–557, Apr. 1997.

[4] C. S. Lent, B. Isaksen, and M. Lieberman, "Molecular quantum-dot cellular automata," *J. Amer. Chem. Soc.*, vol. 125, no. 4, pp. 1056–1063, Jan. 2003.

[5] K. Hennessy and C. S. Lent, "Clocking of molecular quantum-dot cellular automata," *J. Vac. Sci. Technol. B, Microelectron. Process. Phenom.*, vol. 19, no. 5, pp. 1752–1755, Sep. 2001.

[6] T. Oya, T. Asai, T. Fukui, and Y. Amemiya, "A majority-logic nanodevice using a balanced pair of single-electron boxes," *J. Nanosci. Nanotech.*, vol. 2, no. 3/4, pp. 333–342, Jun.–Aug. 2002.

[7] T. Oya, T. Asai, T. Fukui, and Y. Amemiya, "A majority-logic device using an irreversible single-electron box," *IEEE Trans. Nanotechnol.*, vol. 2, no. 1, pp. 15–22, Mar. 2003.

[8] H. A. Fahmy and R. A. Kiehl, "Complete logic family using tunneling-phase- logic devices," in *Proc. Int. Conf. Microelectron.*, Nov. 1999, pp. 22–24.

[9] S. B. Akers, "Synthesis of combinational logic using three-input majority gates," in *Proc. 3rd Annu. Symp. Switching Circuit Theory Logical Des.*, Oct. 1962, pp. 149–157.

[10] H. S. Miller and R. O. Winder, "Majority logic synthesis by geometric methods," *IRE Trans. Electron. Comput.*, vol. EC-11, no. 1, pp. 89–90, Feb. 1962.

[11] S. Muroga, *Threshold Logic and its Applications*. New York: Wiley, 1971.

[12] R. Zhang, K. Walus, W. Wang, and G. A. Jullien, "A method of majority logic reduction for quantum cellular automata," *IEEE Trans. Nanotechnol.*, vol. 3, no. 4, pp. 443–450, Dec. 2004.

[13] K. Walus, G. Schulhof, G. A. Jullien, R. Zhang, and W. Wang, "Circuit design based on majority gates for applications with quantum-dot cellular automata," in *Proc. IEEE Asilomar Conf. Signals, Syst., Comput.*, Nov. 2004, pp. 1354–1357.

[14] M. Bonyadi, S. Azghadi, N. Rad, K. Navi, and E. Afjei, "Logic optimization for majority gate-based nanoelectronic circuits based on genetic algorithm," in *Proc. Int. Conf. Elec. Eng. (ICEE)*, Apr. 2007, pp. 1–5.

[15] S. Rai, "Majority gate based design for combinational quantum cellular automata (QCA) circuits," in *Proc. 40th Southeastern Symp. Syst. Theory*, Mar. 2008, pp. 222–224.

[16] R. Zhang, P. Gupta, and N. K. Jha, "Majority and minority network synthesis with application to QCA-, SET-, and TPL-based nanotechnologies," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 26, no. 7, pp. 1233–1245, Jul. 2007.

[17] Z. Huo, Q. Zhang, S. Haruehanroengra, and W. Wang, "Logic optimization for majority gate-based nanoelectronic circuits," in *Proc. Int. Symp. Circuits Syst. (ISCAS)*, May 2006, pp. 1307–1310.

[18] M. Crocker, X. Hu, M. Niemier, M. Yan, and G. Bernstein, "PLAs in quantum-dot cellular automata," *IEEE Trans. Nanotechnol.*, vol. 7, no. 3, pp. 376–386, May 2008.

[19] K. Kim, K. Wu, and R. Karri, "The robust QCA adder designs using composable QCA building blocks," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 26, no. 1, pp. 176–183, Jan. 2007.

[20] G. De Micheli, *Synthesis and Optimization of Digital Circuits*. New York: McGraw-Hill, 1994.

[21] G. D. Hachtel and F. Somenzi, *Logic Synthesis and Verification Algorithms*. Norwell, MA: Kluwer, 1998.

[22] E. M. Sentovich *et al.*, "SIS: A system of sequential circuit synthesis," UC Berkeley, Berkeley, CA, Electron. Res. Lab. Memo, No. UCB/ERL M92/41, May 1992.

[23] R. Lisanke, "Logic synthesis and optimization benchmarks," Microelectron. Center North Carolina, Research Triangle Park, NC, Tech. Rep., 1988.

[24] K. Walus, T. Dysart, G. Jullien, and R. Budiman, "QCADesigner: A rapid design and simulation tool for quantum-dot cellular automata," *IEEE Trans. Nanotechnol.*, vol. 3, no. 1, pp. 26–29, Mar. 2004.

[25] QCADesigner Documentation. [Online]. Available: http://www.qcadesigner.ca
[26] P. Gupta, N. K. Jha, and L. Lingappan, "A test generation framework for quantum cellular automata circuits," *IEEE Trans. VLSI Syst.*, vol. 15, no. 1, pp. 24–36, Jan. 2007.

Kun Kong received the B.S. degree in mathematics from Shandong University, Shandong, China, in 2003. He is currently working toward the Ph.D. degree at the Institute of Matthematics, Academy of Mathematics and Systems Science, Chinese Academy of Science, Beijing.

His current research interests include nanotechnology, especially quantum-dot cellular automata, computer-aided design tools, and quantum computation.

Yun Shang received the B.S. degree in mathematics from Shandong Normal University, Shandong, China, in 1996, and the M.S. degree in computer science and the Ph.D degree in quantum logic from Shaanxi Normal university, Shaanxi, China, in 2002 and 2005, respectively.

From 2005 to 2007, she worked as a postdoctoral at the Academy of Mathematics and Systems Science, Chinese Academy of Science, Beijing, where she had major in quantum information and computer science. Since 2007, she has been a Lecturer at the Institute of Matthematics, Academy of Mathematics and Systems Science, Chinese Academy of Science,. Her current research interests include computer-aided design for quantum dot cellular automata, quantum computation, quantum logic, and formal semantics.

Ruqian Lu received the Diploma degree in mathematics from Jena University, Jena, Germany, in 1959.

Currently, he is a Professor at the Institute of Mathematics, Academy of Mathematics and Systems Science, Chinese Academy of Sciences, Beijing, China. He is also with the following organizations: Shanghai Key Lab of Intelligent Information Processing, Department of Computer Science and Engineering, Fudan University, Shanghai; the Key Lab of Intelligent Information Processing, Institute of Computing Technology, Chinese Academy of Sciences; Beijing University of Technology and Beijing Key Lab of Multimedia and Intelligent Software, Beijing; the Key Lab of High Confidence Software, Peking University, Beijing, as a Professor. His research interests include quantum information processing, formal semantics, artificial intelligence, knowledge engineering, knowledge-based software engineering, bioinformatics, etc. He is the author of more than 100 papers and several books in these fields.

Mr. Lu is a Fellow of the Chinese Academy of Sciences.