

# A multi-objective synthesis methodology for majority/minority logic networks

Moein Sarvaghad-Moghaddam<sup>1</sup> · Ali A. Orouji<sup>1</sup>  · Monireh Houshmand<sup>2</sup>

Published online: 29 November 2016  
© Springer Science+Business Media New York 2016

**Abstract** New technologies such as quantum-dot cellular automata, single-electron tunneling, tunneling phase logic, and all-spin logic devices have been widely advocated in nanotechnology as a response to the physical limits associated with complementary metal–oxide–semiconductor technology at atomic scales. Some of their peculiar features are their smaller size, higher speed, higher switching frequency, lower power consumption, and higher scale integration. In these technologies, the majority (or minority) and inverter gates are employed for the production of the functions as this set of gates makes a universal set of Boolean primitives in these technologies. An important step in the generation of Boolean functions using the majority gate is reducing the number of involved gates. In this paper, a multi-objective synthesis methodology (with the objective priority of gate counts, gate levels, and the number of inverter gates) is presented for finding the minimal number of possible majority gates in the synthesis of Boolean functions using the proposed majority specification matrix (MSM) concept. Moreover, based on MSM, a synthesis flow is proposed for the synthesis of multi-output Boolean functions. To reveal the efficiency of the proposed method, it is compared with a meta-heuristic method, multi-objective genetic programming (GP). Besides, it is applied to synthesize MCNC benchmark circuits. The results are indicative of the outperformance of the proposed method in comparison to the multi-objective GP method. Also, for the MCNC benchmark circuits, there is an average reduction of 10.5% in the number of levels as well as

16.8% and 33.5% in the number of majority and inverter gates, respectively, as compared to the best available method.

**Keywords** Logic synthesis · Majority gates · Quantum-dot cellular automata (QCA) · Multi-objective

## 1 Introduction

The requirements for increasing speed and decreasing power have led to scaling of feature sizes in complementary metal–oxide–semiconductor (CMOS) technology. More scaling of feature sizes is not possible due to physical limits such as quantum effects and non-deterministic behavior of small currents [1]. Hence, in response to the mentioned limitations, a number of other methods such as quantum-dot cellular automata (QCA) [2], single-electron tunneling (SET) [3,4], tunneling phase logic (TPL) [5], and spin-based devices [6–9] can be used as possible alternatives to CMOS.

QCA was first proposed by Lent [10,11]. QCA is a promising transistor-less technology and beyond-CMOS technology and will play a crucial role in the future of supercomputing [12,13]. The fundamental unit of QCA is a QCA cell which is composed of four dots located at the corners of a square. This technology acts on the basis of Coulombic interactions of electrons trapped in quantum dots. In QCA, the three-input majority and inverter gates are the fundamental primitives.

All-spin logic (ASL) [14] nanotechnology also implements majority logic gates. The fundamental logical device in TPL is a minority gate [5], which is the complement of majority logic. The minority logic synthesis problem is analogous to the majority logic synthesis problem. SET implements both majority and minority logic [3,4].

✉ Ali A. Orouji  
aliaorouji@semnan.ac.ir

<sup>1</sup> Department of Electrical and Computer Engineering, Semnan University, Semnan, Iran

<sup>2</sup> Department of Electrical Engineering, Imam Reza International University, Mashhad, Iran

In CMOS technology, “NAND/NOR/inverter” gates are used to implement circuits; thus, methods created for the synthesis of functions such as Karnaugh maps (K-maps), which produce simplified expressions in the two standard forms named as sum of product (SOP) and product of sum, are not efficient enough for the synthesis of functions to present the simplest possible form for the QCA technology.

Some of researchers [15–18] have proposed effective solutions to the synthesis of QCA-based logic structures. However, these methods were only suitable for small networks as they were used to manually solve the problems or synthesize three-input functions. Synthesis methods proposed in [16, 19] are based on a geometrical interpretation of only three-variable Boolean functions to reduce the majority expressions created by SOPs. These methods led to the creation of thirteen standard functions, which are used for synthesis of the three-input functions. Huo et al. in their study [20] have introduced a table consisting of twenty standard functions and their corresponding majority expressions. First, a given Boolean function is simplified to a function presented in the mentioned table, and then, as a result, a majority expression equivalent to this table is chosen. Some methods [21–24] have applied meta-heuristic algorithms such as genetic algorithm (GA) and genetic programming (GP) for simplification of logic functions. Bonyadi et al. [21] used GA for optimization of a given single-output Boolean function by majority and inverter gates, while Houshmand et al. in [22, 23] applied GP algorithm for optimization of multi-output functions. In [24], the work proposed in [21] has been extended, and a multi-objective optimization consisting of delay as well as the number of gates has been considered. In [25, 26], by using the standard functions, Boolean functions decomposed to four-feasible networks were converted to their corresponding majority expressions. However, the standard functions obtained in [25, 26] cannot be considered as a complete set. In [27], a full set of standard functions, which is not optimal, was identified according to graph theory.

In this paper, a multi-objective synthesis methodology (with the objective priority of gate counts, gate levels, and the number of NOT gates) is proposed, which can be used for the synthesis of three, four, or higher input functions. The concept of majority specification matrix (MSM) is intro-

duced and employed. Furthermore, the synthesis flow is considered for synthesizing multi-output functions. Since the synthesized majority networks can be trivially converted into minority networks using De Morgan’s theorem, we only focus on majority logic synthesis in this study. To compare the suggested method with the other ones, benchmarks in [24] and MCNC benchmark circuits are used. This approach results in fewer majority gates and fewer logic levels as compared to existing methods [28, 29]. The resulting majority/minority network can then be used in ASL-, QCA-, TPL-, or SET-based nanotechnologies.

The rest of the paper is organized as follows: In Sect. 2, some related background materials are presented. Section 3 introduces the proposed method in detail. In Sect. 4, a synthesis flow for multi-output functions is introduced. Section 5 presents the results and finally Sect. 6 concludes the paper.

## 2 Background material

In this section, basic concepts in QCA technology such as QCA and QCA devices are explained.

### 2.1 Quantum-dot cellular automata

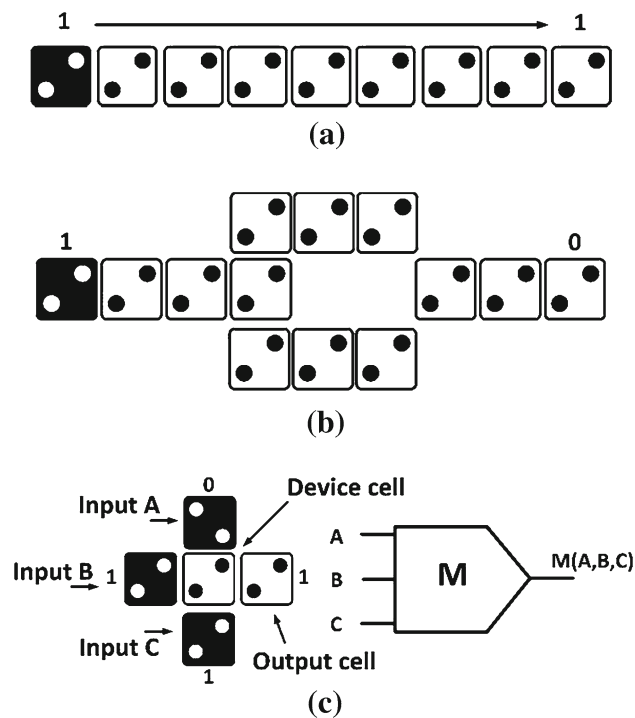
A standard QCA cell (Fig. 1a) is composed of four dots located at the corners of a square. Two free electrons can tunnel to any quantum dot within the cell [30]. Because of Coulombic interactions, the electrons occupy diagonally opposite positions. Depending on the position of the cell, polarization of a QCA cell can be determined with two stable cell polarization states as shown in Fig. 1b. These configurations are denoted as cell polarization  $P = +1$  (binary ‘1’ state) and  $P = -1$  (binary ‘0’ state).

### 2.2 QCA devices

In this section, the basic devices used in QCA such as QCA wires, QCA inverters, and QCA majority voters will be introduced. In a QCA wire, a binary signal propagates from input to output because of the Coulombic interactions between cells. In a QCA inverter, cells oriented at  $45^\circ$  to each other take on opposing polarization. A QCA majority gate can per-

**Fig. 1** **a** Structure of a QCA cell with four quantum dots. **b** QCA cell with two different polarizations





**Fig. 2** Representation of **a** QCA wire, **b** inverter gate, and **c** QCA majority gate

form a three-input majority gate. Equation (1) presents the logic function of a three-input majority gate where  $A$ ,  $B$ , and  $C$  are the three inputs.

$$M(A, B, C) = AB + BC + CA. \quad (1)$$

By forcing one of the three inputs of the majority gate to a constant logic “0” or a “1,” the majority gate can be used to perform AND/OR operations as shown in the following equations:

$$M(A, B, 0) = AB, \quad M(A, B, 1) = A + B. \quad (2)$$

Figure 2 demonstrates a QCA wire (a), inverter gate (b), and majority gate (c).

### 2.3 Spin devices

All-spin logic is a low-power switch with switching mechanism based on spin torque. In these circuits, the input and output are in the electrical domain, while the processing within the circuit happens in the spin domain. Figure 3a shows the layout of an ASL device which has four terminals: (a) terminal1: VDD, (b) terminal2: VSS, (c) terminal3: input, and (d) terminal4: output. The device is composed of one magnet (or nanomagnet), which is the information storage unit, one high-polarization ( $\text{High}_p$ ) magnet–channel interface for the input, one low-polarization ( $\text{Low}_p$ ) magnet–channel interface for

the output, an isolation between receiving (input) and transmitting (output) sides and spin channels both at receiving and transmitting sides as shown in Fig. 3a. The two stable states of the magnet (left and right spin) are determined by magnetic anisotropy (uniaxial anisotropy,  $K_u$ ) [31]. Also, in Fig. 3b, c the ASL inverter and majority gates are shown, respectively [6].

### 2.4 SET technology

Figure 4a shows a basic minority SET gate. The inputs with three capacitors form a voltage summing system which generates a mean voltage at node  $A$ . If this voltage exceeds a certain threshold, an electron tunnels through single-electron boxes (SEBs) and negates the voltage at  $A$ . Otherwise, the voltage remains positive. Logic 1 and logic 0 are represented by positive and negative voltages, respectively.

A majority gate implemented by a balanced pair of single-electron boxes is shown in Fig. 4b [4]. An electron tunnels through one of the SEBs to make a negative voltage and prevents movements of other electron when VDD increases. Hence, the stable voltage states for the two SEBs are (1, 0) and (0, 1) based on the inputs. For example, if all inputs are 0, the voltage state is (0, 1) and node  $B$  has a negative voltage.

By fixing one of the three inputs to logic 0 or 1, “NAND” gate and “NOR” gate are achieved for SET minority gate, while “AND” gate and NOR gate are obtained for SET majority gate, respectively [3].

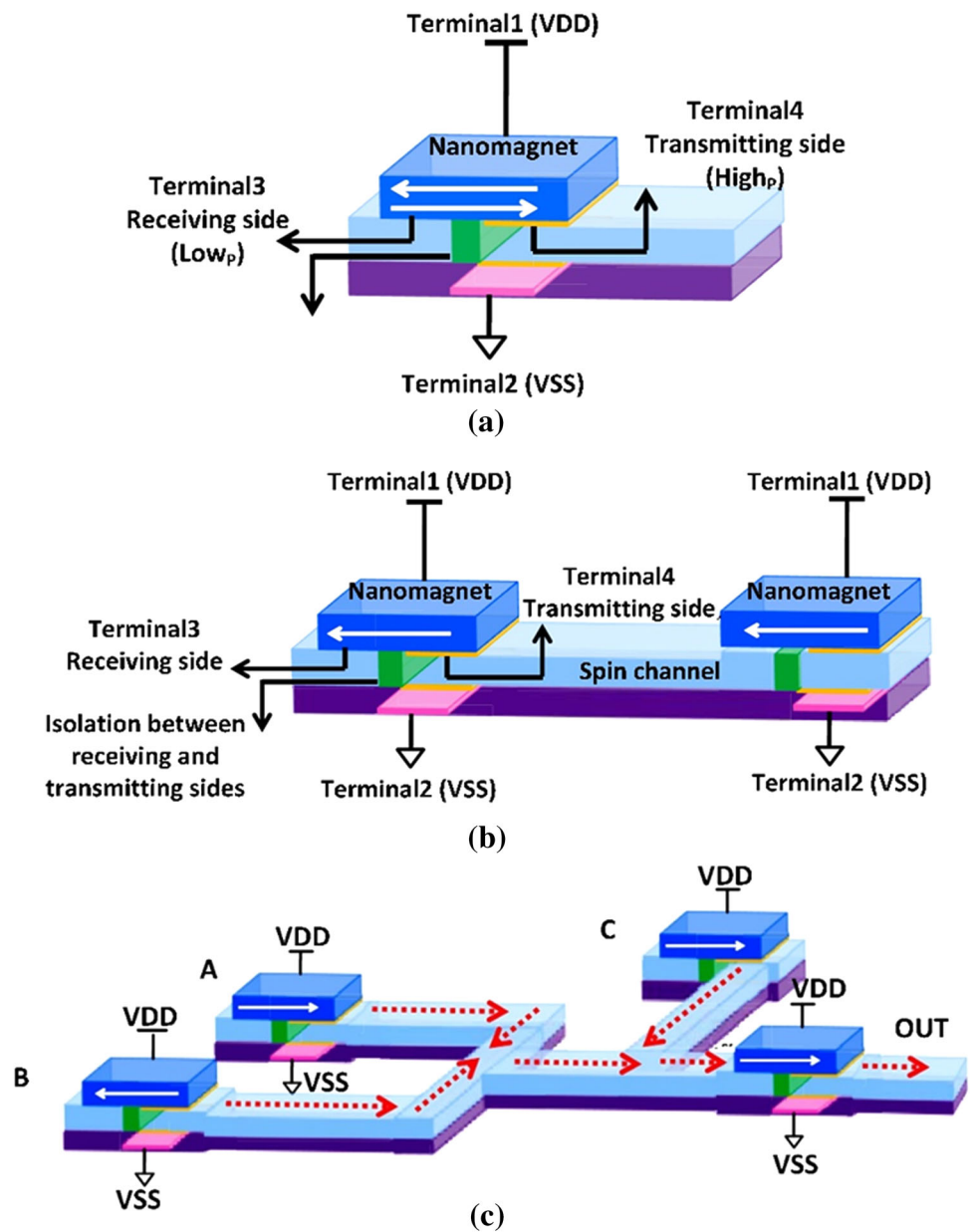
### 2.5 TPL technology

Figure 4c shows a minority gate in TPL that uses two phases. The phase of a waveform is used in TPL to represent logic values in digital circuits.  $C_j$  indicates the tunneling junction capacitance. The operation of TPL is based on the phase locking of SET oscillations to a pump signal that is distributed throughout the circuit. Since the pump frequency is set to twice the tunneling frequency, the electrical phase of the locked oscillation can have two different values. Each value represents a binary encoding [5,32]. A TPL minority gate can be a two-input NOR or NAND gate by fixing one of its inputs to logic high or logic low, respectively.

## 3 Proposed method

The suggested synthesis method is based on the creation of MSM. In the mentioned matrix, all of the input states of a majority function are placed in columns of this matrix. In fact, the specification of the majority function output for each certain input state ( $a, b, c$ ) is placed in each of MSM columns. As there are  $2^3$  possible input states for a three-input majority gate as shown in Fig. 5, the dimensions of the matrix are

**Fig. 3** **a** Layout of an ASL device. **b** Layout of inverter using ASL\_NC. **c** ASL majority gate M3 (direction of information propagation is shown with *arrows*) [6]



$8 \times 8$ , and it can be considered as a regular matrix. More details on this topic can be found in [33–35]. In this matrix, binary number of each column is related to a certain majority function, for instance, the binary number of the second column is 001 which is equivalent to  $\text{Maj}(a', b', c)$ ; it means that zeros in the binary number of each column are related to an inverter gate in the input of majority gate with a certain order.

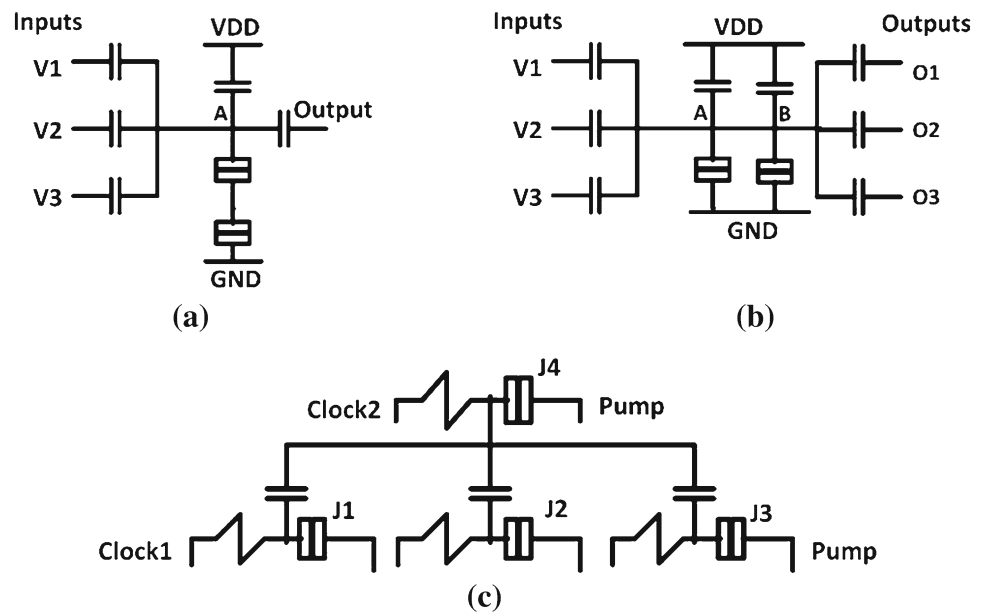
The following features can be specified in the mentioned matrix:

- Specification of output of pair columns (4, 5), (3, 6), (2, 7), and (1, 8) are complementary.

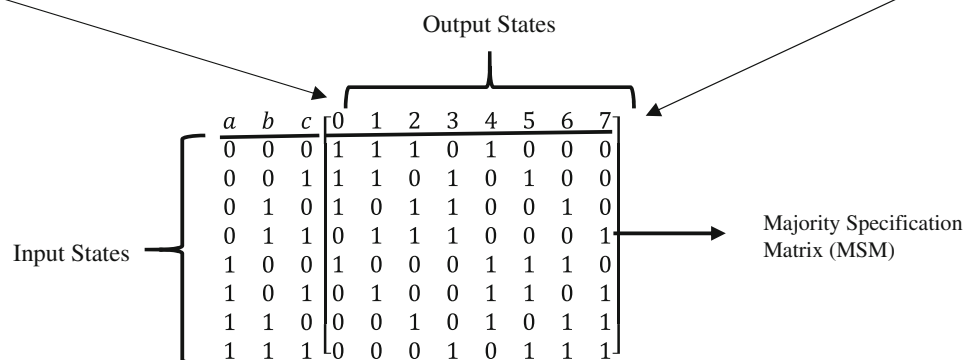
- With respect to each of the two non-complementary columns, there are exactly two input states with the value of one, which are common in each of the two columns.
- If two majority gates are common in two input variables, then the following properties hold:

$$\begin{aligned} \text{Maj}(a, b, c) + \text{Maj}(a, b, c') &= \text{Maj}(a, b, c + c') \\ &= \text{Maj}(a, b, 1), \\ \text{Maj}(a, b, c) \times \text{Maj}(a, b, c') &= \text{Maj}(a, b, c \times c') \\ &= \text{Maj}(a, b, 0). \end{aligned} \quad (3)$$

**Fig. 4** a SET minority gate. b SET majority gate. c TPL minority gate



| Maj(a',b',c') | Maj(a',b',c) | Maj(a',b,c') | Maj(a',b,c) | Maj(a,b',c') | Maj(a,b',c) | Maj(a,b,c') | Maj(a,b,c) |
|---------------|--------------|--------------|-------------|--------------|-------------|-------------|------------|
| 000           | 001          | 010          | 011         | 100          | 101         | 110         | 111        |
| 0             | 1            | 2            | 3           | 4            | 5           | 6           | 7          |



**Fig. 5** Structure representation of majority specification matrix (MSM)

- Changing the order of input variables does not change the specification function in each column.

In the following sections, the application of MSM for the synthesis of Boolean functions will be elaborated on.

### 3.1 Three-input Boolean functions

This section explains the application of MSM for the synthesis of three-input Boolean logic functions. To this end, the proposed methods are divided into two basic parts and a post-processing method. These methods have been generally designed to achieve the following two objectives:

- First, the simplest expression based on majority function should be achieved for each function.

- The number of common expressions in the outputs of multi-output functions should be the maximum possible number and the minimum number of inverter gates should exist in them.

#### 3.1.1 Base of Method 1

In the first method, one majority gate and AND/OR functions are used. First, the specification function is compared to each column of MSM. One of the columns with the most identical number of ones is selected, i.e., Hamming code created between columns of MSM with the given specification function had the minimum possible number of ones. Then, through the application of AND and OR functions, additional ones are removed, and minterms with additional zeros

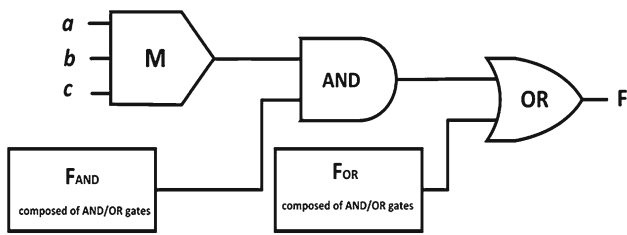


Fig. 6 An overall schematic of Method 1

Table 1 Impacts of applied AND function (applied between selected majority gate and  $F_{AND}$ ) and OR function (applied between majority gate,  $F_{AND}$  and  $F_{OR}$ ), respectively

| Output function | Majority gate selected | $F_{AND}$ | $F_{OR}$ |
|-----------------|------------------------|-----------|----------|
| 0               | 1                      | 0         | 0        |
| 0               | 0                      | X         | 0        |
| 1               | 0                      | X         | 1        |
| 1               | 1                      | 1         | X        |

are converted to ones, respectively. Following this step, for each part K-map is used for further simplification of the final expression. In Fig. 6, an overall schematic of Method 1 is shown.

The impacts of AND and OR functions on the main function output and selected majority gate are presented in Table 1. In this table, the impact of AND gate applied between selected majority gate and  $F_{AND}$  and then OR gate applied between majority gate and  $F_{OR}$  are shown. As AND and OR gates are complementary in their usage, the order of using them is not important. In Method 1, first AND and then OR gates are applied, respectively.

In this table, X denotes “don’t care” state. Moreover, the majority expressions for AND and OR gates are as follows:

$$AND = Maj(f_1, f_2, 0), \quad OR = Maj(f_1, f_2, 1). \quad (4)$$

In these equations,  $f_1$  and  $f_2$  functions are obtained from Method 1. The following example explains this method in more detail:

Example 1 Consider specification  $F(a, b, c) = (0, 3, 6)$ (the numbers represent minterms contained in the function) defined as the first and the second columns of Table 2.

As shown in Table 2, Column 2 of MSM is selected as the most similar column to the specification function ( $M(a', b', c')$ ). There is an additional minterm, in which function value is 1, i.e., 010. By the application of AND function as shown in Column 4 of Table 2, additional one value is converted to zero value. In Table 2, AND operation must be applied between Columns 3 and 4. It must be noted that Column 4 is created using Table 1. For simplification of the function in Column 4, K-map is used as shown in Table 3.

Table 2 Representation of the specification function related to Example 1 by the application of the proposed Method 1

| $a$ | $b$ | $c$ | $F$ | $F(2) = M(a', b, c')$ | $F_{AND}$ |
|-----|-----|-----|-----|-----------------------|-----------|
| 0   | 0   | 0   | 1   | 1                     | 1         |
| 0   | 0   | 1   | 0   | 0                     | X         |
| 0   | 1   | 0   | 0   | 1                     | 0         |
| 0   | 1   | 1   | 1   | 1                     | 1         |
| 1   | 0   | 0   | 0   | 0                     | X         |
| 1   | 0   | 1   | 0   | 0                     | X         |
| 1   | 1   | 0   | 1   | 1                     | 1         |
| 1   | 1   | 1   | 0   | 0                     | X         |

Table 3 Simplification of  $F_{AND}$  created by K-map

| $a$ | $bc$  |       |       |    |
|-----|-------|-------|-------|----|
|     | 00    | 01    | 11    | 10 |
| 0   | 1     | X = 1 | 1     | 0  |
| 1   | X = 1 | X = 1 | X = 1 | 1  |

As presented in Table 3, for further simplification of function, all “don’t care” states receive a value of one. Hence, one possible majority expression for  $F_{AND}$  can be obtained as follows:

$$F_{AND} = a + b' + c = Maj(a, Maj(b', c, 1), 1). \quad (5)$$

The final result is provided as follows:

$$F = Maj(Maj(a', b, c'), F_{AND}, 0) = Maj(Maj(a', b, c') Maj(a, Maj(b', c, 1), 1), 0). \quad (6)$$

3.1.1.1 Another structure of Method 1 It is worth mentioning that instead of specifying the most similar column from MSM, it is better to compare the columns of function input ( $a, b, c$ ) with the specification of function output. It is due to this issue that in some functions, the places of ones in input columns are most similar to those of output columns. Hence, one of the majority gates would be removed. This method can also be used for the proposed methods in the next sections.

### 3.1.2 Base of Method 2

In the second method, a majority gate with three inputs  $f_1, f_2, f_3$  is employed ( $F = Maj(f_1, f_2, f_3)$ ); then, expressions related to the three functions are obtained. In Fig. 7, an overall schematic of Method 2 is shown.

In this method, similar to Method 1, first, the column of MSM which is the most similar to the function specification is selected ( $f_1$ ), and then the value of function  $f_2$  is obtained according to Table 4.

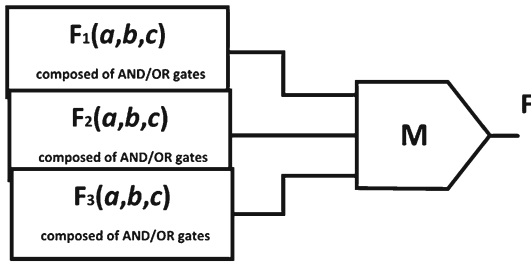


Fig. 7 An overall schematic of Method 2

Table 4 Obtaining  $f_2$  function according to  $f_1$  and main function

| Main function | $f_1 = \text{Majority gate selected}$ | $f_2$ |
|---------------|---------------------------------------|-------|
| 0             | 0                                     | X     |
| 0             | 1                                     | 0     |
| 1             | 0                                     | 1     |
| 1             | 1                                     | X     |

Table 5 Representation of specification function related to Example 2 after the application of Method 2

| a | b | c | F | $f_1 = \text{Maj}(a', b', c')$ | $f_2$ | $f_3$ |
|---|---|---|---|--------------------------------|-------|-------|
| 0 | 0 | 0 | 0 | 1                              | 0     | 0     |
| 0 | 0 | 1 | 1 | 1                              | X = 0 | 1     |
| 0 | 1 | 0 | 1 | 1                              | X = 0 | 1     |
| 0 | 1 | 1 | 0 | 0                              | X = 0 | X = 1 |
| 1 | 0 | 0 | 1 | 1                              | X = 1 | X = 0 |
| 1 | 0 | 1 | 1 | 0                              | 1     | 1     |
| 1 | 1 | 0 | 1 | 0                              | 1     | 1     |
| 1 | 1 | 1 | 1 | 0                              | 1     | 1     |

As demonstrated in Table 4, in the process of comparing the main function and the selected majority gate, if the minterm has hamming code one, then the value of function  $f_2$  must be equal to the value of main function; otherwise, the value of minterm in function  $f_2$  should be “don’t care” (X). The mentioned point is due to the feature of majority function. If the number of ones in a minterm is greater than or equal to two, then the value of output of majority function should be one; otherwise it should be zero. Then, for further simplification of function  $f_2$ , K-map is used. When the value of “don’t care” states is determined considering the above-stated feature for majority gate, the value of minterms of function  $f_3$  can be obtained considering Table 4. The following example illustrates the idea:

Example 2 Consider specification  $F(a, b, c) = (1, 2, 4, 5, 6, 7)$  defined in the second column of Table 5.

First, the column of MSM with the least difference with the main function is selected ( $f_1 = \text{Maj}(a', b', c')$ ). Then, the

Table 6 Representation of K-map applied for simplification of function  $f_2$

| bc \ a | 00    | 01    | 11    | 10    |
|--------|-------|-------|-------|-------|
| 0      | 0     | X = 0 | X = 0 | X = 0 |
| 1      | X = 1 | 1     | 1     | 1     |

Table 7 Applying K-map for simplification of function  $f_3$

| a \ bc | 00    | 01 | 11    | 10 |
|--------|-------|----|-------|----|
| 0      | 0     | 1  | X = 1 | 1  |
| 1      | X = 1 | 1  | 1     | 1  |

value of function  $f_2$  is obtained according to Table 4 and K-map presented in Table 6 as follows ( $f_2 = a$ ).

The value of function  $f_3$  is obtained by considering  $f_2$  and Table 4; for example, the value obtained for  $f_2$  in state (001) is zero, which is not equivalent to the values of  $f_1$  and main function. Hence, the value of  $f_3$  would be equal to the value of main function. Furthermore, the values of  $f_2$ ,  $f_1$ , and the main function in state (011) are the same; therefore, it can be stated that the value of  $f_3$  is “don’t care.” For further simplification of  $f_3$ , K-map is used as presented in Table 7. The logic expression for  $f_3$  is as follows:

$$f_3 = (b + c) = \text{Maj}(b, c, 1). \tag{7}$$

The total logic expression for main function (F) is

$$F = \text{Maj}(\text{Maj}(a', b', c')a, \text{Maj}(b, c, 1)). \tag{8}$$

3.1.2.1 Another structure of Method 2 For some functions, a combination of the first (AND, OR) and the second methods can lead to better results. In the following example, the above-mentioned method is explained.

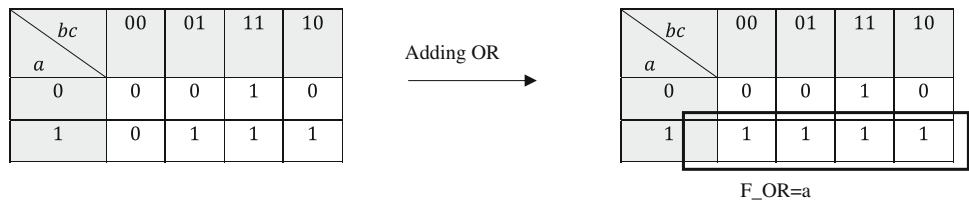
Example 3 Consider function  $F(a, b, c) = (3, 4, 6)$  as shown in Table 8.

First, the most similar column of MSM to the main function is selected ( $f_1 = \text{Maj}(a, b, c)$ ). By selecting function  $f_1$  as presented in Table 8 and the examination of K-map, it can be observed that in order to apply the OR operation between  $f_1$  and  $F_{\text{OR}}$  (Columns 3 and 4 in Table 8), all of the needed 1’s states should be created without adding extra majority function. It means that the logic expression for  $F_{\text{OR}}$  is ( $F_{\text{OR}} = a$ , as shown in Fig. 8).

**Table 8** An example of the combination of the first and the second methods for improvement of the results

| <i>a</i> | <i>b</i> | <i>c</i> | <i>F</i> | $f_1 = \text{Maj}(7) = \text{Maj}(a, b, c)$ | $F_{OR}$ | $F_{tot}$ | $f_2$   | $f_3$ |
|----------|----------|----------|----------|---|----------|-----------|---------|-------|
| 0        | 0        | 0        | 0        | 0   | 0        | 0         | $X = 0$ | $X$   |
| 0        | 0        | 1        | 0        | 0   | 0        | 0         | $X = 0$ | $X$   |
| 0        | 1        | 0        | 0        | 0   | 0        | 0         | $X = 0$ | $X$   |
| 0        | 1        | 1        | 1        | 1   | 0        | 1         | $X = 0$ | 1     |
| 1        | 0        | 0        | 1        | 0   | 1        | 1         | $X = 0$ | 1     |
| 1        | 0        | 1        | 0        | 1   | 1        | 1         | 0       | 0     |
| 1        | 1        | 0        | 1        | 1   | 1        | 1         | $X = 0$ | 1     |
| 1        | 1        | 1        | 0        | 1   | 1        | 1         | 0       | 0     |

**Fig. 8** Applying OR operation to function  $f_1$



**Table 9** Applying K-map for simplification of function  $f_3$

| <i>a</i> | <i>bc</i> |         |    |         |
|----------|-----------|---------|----|---------|
|          | 00        | 01      | 11 | 10      |
| 0        | $X = 1$   | $X = 1$ | 1  | $X = 1$ |
| 1        | 1         | 0       | 0  | 1       |

The obtained result is shown in Column 5 ( $F_{tot} = \text{Maj}(\text{Maj}(a, b, c), a, 1)$ ). Hence, by consideration of Table 4, comparison of values related to  $F_{tot}$  and the main function, and by the application of K-map, the value of  $f_2$  can be obtained. With respect to the specification function  $f_2$  obtained in Column 6, it is certain that the logic function  $f_2$  is zero ( $f_2 = 0$ ) as it is composed of “don’t care” and “zero” states. Hence, the simplest function with the minimum number of majority gates is the zero function. Moreover, the specification function  $f_3$  is observed to be the same as  $f_2$ . Then, the logic function is simplified by applying K-map (as shown in Table 9). Thus, the logic expression for  $f_3$  is

$$f_3 = a' + c' = \text{Maj}(a', c', 1). \tag{9}$$

As a result, the total logic expression obtained for main function ( $F$ ) is

$$F = \text{Maj}(\text{Maj}(\text{Maj}(a, b, c), a, 1), 0, \text{Maj}(a', c', 1)). \tag{10}$$

### 3.1.3 Post-processing method

To obtain better results, the post-processing method is used, which can be applied to the above-presented methods and can improve the obtained results. In this method, after applying

Method 2 to the specification function and obtaining functions  $f_2$  and  $f_3$ , K-maps related to  $f_2$  and  $f_3$  are studied simultaneously. It is conjectured that if some of 1’s states related to  $f_2$  and  $f_3$  are exchanged, better results would be obtained. The mentioned point is due to the feature of majority function. If the number of ones in a minterm is greater than or equal to two, then the value of output of majority function would be one; otherwise, it would be zero. Furthermore, the following steps are taken into account in this method:

- In the K-maps of  $f_2$  and  $f_3$ , minterms of main function whose corresponding function value is one are marked. Due to the mentioned feature for the majority gate, these places (cubes of K-map) can have numbers of 2 or 3 ones in the majority function (Group 1).
- In the K-maps of  $f_2$  and  $f_3$ , minterms of main function with the zero value of function can have numbers of 0 or 1 ones in the majority function (Group 2).
- Fixed minterms in the K-map  $f_2$  are not considered.
- “Don’t care” states in  $f_2$ , which do not create a square in the K-map and generate an extra state in the K-map related to  $f_3$ , are exchanged in response to the rules of Groups 1 and 2. This method as a search method is continued until the best square in the K-maps is obtained.
- Equation (11) holds for each majority gate. By applying that expression, the number of inverter gates in the general expression can be decreased.

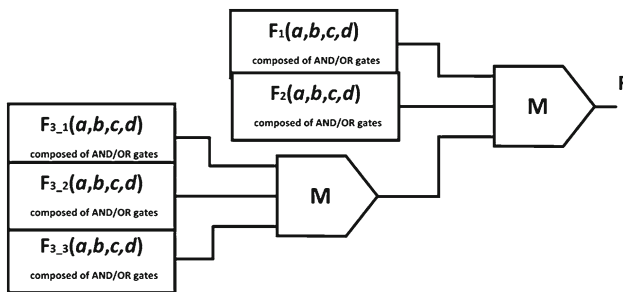
$$\text{Maj}(a', b', c') = \text{Maj}(a, b, c') \tag{11}$$

Generally, only states of  $f_2$ , which are “don’t care,” are considered. Then, the states generated in  $f_3$  which do



**Table 10** Creation of new states in four-input functions according to the three-input functions for converting  $M(b, c, d)$  to  $M(a, b, c)$ . Variable  $d$  is “don’t care”

| Old rows<br>$b c d$ | New permutation of rows<br>$a b c d$ | Old rows<br>$b c d$ | New permutation of rows<br>$a b c d$ |
|---------------------|--------------------------------------|---------------------|--------------------------------------|
| 0 0 0               | 0 0 0 0                              | 1 0 0               | 1 0 0 0                              |
|                     | 0 0 0 1                              |                     | 1 0 0 1                              |
| 0 0 1               | 0 0 1 0                              | 1 0 1               | 1 0 1 0                              |
|                     | 0 0 1 1                              |                     | 1 0 1 1                              |
| 0 1 0               | 0 1 0 0                              | 1 1 0               | 1 1 0 0                              |
|                     | 0 1 0 1                              |                     | 1 1 0 1                              |
| 0 1 1               | 0 1 1 0                              | 1 1 1               | 1 1 1 0                              |
|                     | 0 1 1 1                              |                     | 1 1 1 1                              |



**Fig. 9** An overall schematic of Method 3

not make a square in  $f_2$  are taken into account, and these states are exchanged between  $f_2$  and  $f_3$  to make squares in  $f_2$  and  $f_3$ . Then, the squares providing the least number of majority expressions are selected from the overall obtained squares.

- **Proposition:** Suppose  $f, g, u,$  and  $d$  are Boolean functions and the following expression exists between them:  $y = u(fg' + gd)$ , where  $g'$  is the complement of  $g$ . Then, the equivalent majority expression is as follows:  $y = M(u, M(f, g', 0) M(g, d, 0))$ .

*Proof:* By extending equivalent majority expression, we have

$$y = u(fg' + gd) + ((fg') \times (gd)) = u(fg' + gd).$$

### 3.2 Four- and higher-input Boolean functions

In this section, for the synthesis of four- and higher-input Boolean functions, it must be considered that the majority gate has three inputs; thus, for instance, the methods are explained for four-input functions. Accordingly, the methods presented in this section can be generalized to higher inputs.

Taking into account that the majority gate has three inputs; thus, for four-input functions with consideration of the inputs

**Table 11** Specification function used in Example 4 and calculation of functions  $f_2$  and  $f_3$

| $a$ | $b$ | $c$ | $d$ | $F$ | $f_1 = M(7) = M(a, b, c)$ | $f_2$   | $f_3$   |
|-----|-----|-----|-----|-----|---------------------------|---------|---------|
| 0   | 0   | 0   | 0   | 0   | 0                         | $X = 0$ | $X = 1$ |
| 0   | 0   | 0   | 1   | 0   | 0                         | $X = 0$ | $X = 1$ |
| 0   | 0   | 1   | 0   | 0   | 0                         | $X = 0$ | $X = 1$ |
| 0   | 0   | 1   | 1   | 0   | 0                         | $X = 0$ | $X = 1$ |
| 0   | 1   | 0   | 0   | 0   | 0                         | $X = 0$ | $X = 0$ |
| 0   | 1   | 0   | 1   | 0   | 1                         | 0       | 0       |
| 0   | 1   | 1   | 0   | 0   | 0                         | $X = 0$ | $X = 1$ |
| 0   | 1   | 1   | 1   | 0   | 1                         | 0       | 0       |
| 1   | 0   | 0   | 0   | 0   | 0                         | $X = 0$ | $X = 1$ |
| 1   | 0   | 0   | 1   | 1   | 1                         | $X = 0$ | 1       |
| 1   | 0   | 1   | 0   | 0   | 0                         | $X = 0$ | $X = 1$ |
| 1   | 0   | 1   | 1   | 1   | 1                         | $X = 0$ | 1       |
| 1   | 1   | 0   | 0   | 0   | 1                         | 0       | 0       |
| 1   | 1   | 0   | 1   | 0   | 1                         | 0       | 0       |
| 1   | 1   | 1   | 0   | 1   | 1                         | $X = 0$ | 1       |
| 1   | 1   | 1   | 1   | 0   | 1                         | 0       | 0       |

**Table 12** Applying K-map for obtaining function  $f_3$

| $ab$ | $cd$    |         |         |         |
|------|---------|---------|---------|---------|
|      | 00      | 01      | 11      | 10      |
| 00   | $X = 1$ | $X = 1$ | $X = 1$ | $X = 1$ |
| 01   | $X = 0$ | 0       | 0       | $X = 1$ |
| 11   | 0       | 0       | 0       | 1       |
| 10   | $X = 1$ | 1       | 1       | $X = 1$ |

intended to function, there would be four permutations ( $C(3, 4) = 4$ , where  $C$  denotes the combination of permutations.), as the order of inputs does not matter. It is assumed that the inputs of the main function are  $a, b, c,$  and  $d$ . Moreover, it must be mentioned that input ( $a$ ) is the most significant one and input ( $d$ ) is the least significant one.

$$\text{Maj}(b, c, d), \text{Maj}(a, b, c, ), \text{Maj}(a, c, d), \text{Maj}(a, b, d) \quad (12)$$

**Fig. 10** **a** Applying K-map for function  $f_2$  and **b** applying K-map for function  $f_3$ . Applying post-processing method to  $f_2$  and  $f_3$

|                    |         |         |         |         |           |
|--------------------|---------|---------|---------|---------|-----------|
| $cd \backslash ab$ | 00      | 01      | 11      | 10      |           |
| 00                 | $X = 0$ | $X = 0$ | $X = 0$ | $X = 0$ | } $X = 1$ |
| 01                 | $X = 0$ | 0       | 0       | $X = 0$ |           |
| 11                 | 0       | 0       | 0       | $X = 0$ |           |
| 10                 | $X = 0$ | $X = 0$ | $X = 0$ | $X = 0$ |           |

(a)

|                    |         |         |         |         |           |
|--------------------|---------|---------|---------|---------|-----------|
| $cd \backslash ab$ | 00      | 01      | 11      | 10      |           |
| 00                 | $X = 1$ | $X = 1$ | $X = 1$ | $X = 1$ | } $X = 0$ |
| 01                 | $X = 0$ | 0       | 0       | $X = 1$ |           |
| 11                 | 0       | 0       | 0       | 1       |           |
| 10                 | $X = 1$ | 1       | 1       | $X = 1$ |           |

$X = 0$

(b)

**Table 13** Specification function used in Exp. 5 and calculation of  $f_2$  and  $f_3$

| A | B | C | D | F | $f_1 = M(0, c, d)$ | $f_2$   | $f_3$ |
|---|---|---|---|---|--------------------|---------|-------|
| 0 | 0 | 0 | 0 | 0 | 0                  | $X = 0$ | $X$   |
| 0 | 0 | 0 | 1 | 0 | 0                  | $X = 0$ | $X$   |
| 0 | 0 | 1 | 0 | 0 | 0                  | $X = 0$ | $X$   |
| 0 | 0 | 1 | 1 | 1 | 1                  | $X = 0$ | 1     |
| 0 | 1 | 0 | 0 | 1 | 0                  | 1       | 1     |
| 0 | 1 | 0 | 1 | 0 | 0                  | $X = 1$ | 0     |
| 0 | 1 | 1 | 0 | 0 | 0                  | $X = 1$ | 0     |
| 0 | 1 | 1 | 1 | 1 | 1                  | $X = 1$ | $X$   |
| 1 | 0 | 0 | 0 | 0 | 0                  | $X = 0$ | $X$   |
| 1 | 0 | 0 | 1 | 0 | 0                  | $X = 0$ | $X$   |
| 1 | 0 | 1 | 0 | 0 | 0                  | $X = 0$ | $X$   |
| 1 | 0 | 1 | 1 | 0 | 1                  | 0       | 0     |
| 1 | 1 | 0 | 0 | 0 | 0                  | $X = 1$ | 0     |
| 1 | 1 | 0 | 1 | 0 | 0                  | $X = 1$ | 0     |
| 1 | 1 | 1 | 0 | 0 | 0                  | $X = 1$ | 0     |
| 1 | 1 | 1 | 1 | 1 | 1                  | $X = 1$ | $X$   |

**Table 14** K-map used for calculation  $f_2$

|                    |         |         |         |         |
|--------------------|---------|---------|---------|---------|
| $ab \backslash cd$ | 00      | 01      | 11      | 10      |
| 00                 | $X = 0$ | $X = 0$ | $X = 0$ | $X = 0$ |
| 01                 | 1       | $X = 1$ | $X = 1$ | $X = 1$ |
| 11                 | $X = 1$ | $X = 1$ | $X = 1$ | $X = 1$ |
| 10                 | $X = 0$ | $X = 0$ | 0       | $X = 0$ |

the other states, each row of MSM created in Fig. 5 must move to two new rows for four-input functions. For example, consider the  $Maj(a, b, c)$  state. For the creation of the mentioned state, the majority function  $Maj(a, b, c)$  must shift variables  $(a, b, c)$  to the left in comparison to state  $Maj(b, c, d)$ . This method is shown in Table 10. In fact, variable  $d$  in the main permutation  $(b, c, d)$  is as a “don’t care” variable for new permutation  $(a, b, c)$ .

For four-input functions, the following methods are used:

1. In this method, the majority gate is used as a tree expression and employs Method 2 discussed in the three-input functions section. In the mentioned section, if the order of ones in each  $f_2$  or  $f_3$  function leads to the creation of complex specification functions, and as a result many majority gates will be created in each function. Hence, for the synthesis of  $f_2$  or  $f_3$  function, each of them can be considered as a main function. Then, in the process of applying Method 2 explained in three-input functions

For each of the above-mentioned permutations, an MSM must be created according to Fig. 5. For the creation of MSM with  $b, c,$  and  $d$  inputs, two matrices of MSM can be created under each other. It means that two matrices of MSM are placed in one column as cascades. As a result, the matrix of MSM has 16 rows and 8 columns. For the creation of MSM in

**Table 15** Consider function  $f_2$  as the main function for using tree method

| $a$ | $b$ | $c$ | $d$ | $f_3$ | $f_{3,1} = M(a', b', c')$ | $f_{3,2}$ | $f_{3,3}$ |
|-----|-----|-----|-----|-------|---------------------------|-----------|-----------|
| 0   | 0   | 0   | 0   | X     | 1                         | $X = 0$   | $X = 1$   |
| 0   | 0   | 0   | 1   | X     | 1                         | $X = 0$   | $X = 0$   |
| 0   | 0   | 1   | 0   | X     | 1                         | $X = 0$   | $X = 1$   |
| 0   | 0   | 1   | 1   | 1     | 1                         | $X = 0$   | 1         |
| 0   | 1   | 0   | 0   | 1     | 1                         | $X = 0$   | 1         |
| 0   | 1   | 0   | 1   | 0     | 1                         | 0         | 0         |
| 0   | 1   | 1   | 0   | 0     | 0                         | $X = 0$   | $X = 1$   |
| 0   | 1   | 1   | 1   | X     | 0                         | $X = 0$   | $X = 1$   |
| 1   | 0   | 0   | 0   | X     | 1                         | $X = 0$   | $X = 1$   |
| 1   | 0   | 0   | 1   | X     | 1                         | $X = 0$   | $X = 0$   |
| 1   | 0   | 1   | 0   | X     | 0                         | $X = 0$   | $X = 1$   |
| 1   | 0   | 1   | 1   | 0     | 0                         | $X = 0$   | $X = 1$   |
| 1   | 1   | 0   | 0   | 0     | 0                         | $X = 0$   | $X = 1$   |
| 1   | 1   | 0   | 1   | 0     | 0                         | $X = 0$   | $X = 0$   |
| 1   | 1   | 1   | 0   | 0     | 0                         | $X = 0$   | $X = 1$   |
| 1   | 1   | 1   | 1   | X     | 0                         | $X = 0$   | $X = 1$   |

section, the mentioned functions can be synthesized once more. This method can be repeatedly carried out to determine its acceptable level. In Fig. 9, an overall schematic of this method is shown.

- For synthesis of functions with more than three inputs, the methods explained in the previous section can be used.

In the following examples, the above-mentioned methods are explained.

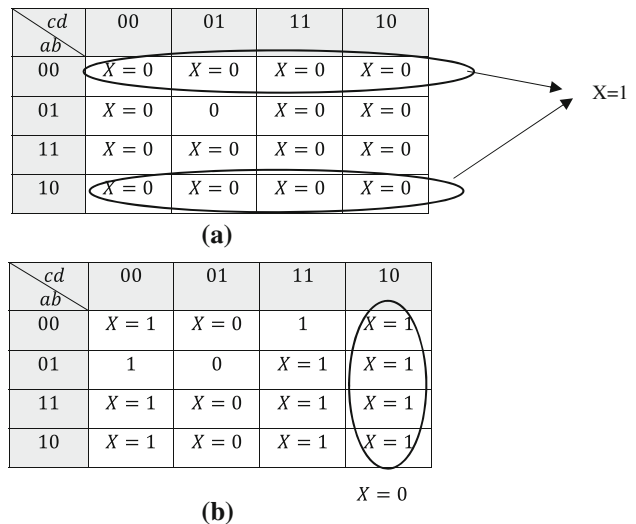
*Example 4* Consider function  $F = (9, 11, 14)$  defined as the first and the second columns of Table 11.

According to Method 2 explained in three-input functions, first, the column of MSM that is the most similar one to the specification function is selected ( $f_1 = M(a, b, d)$ ). Then, specification functions ( $f_2$  and  $f_3$ ) are obtained. It can be certainly stated that the logic function  $f_2$  is zero ( $f_2 = 0$ ) and the logic function  $f_3$  is obtained by the application of K-map as shown in Table 12:

Boolean logic function  $f_3$  is

$$f_3 = b' + cd' = \text{Maj}(b' \text{Maj}(c, d', 0), 1). \tag{13}$$

By the implementation of the post-processing method presented in three-input functions section, the specification functions of  $f_2$  and  $f_3$  can be exchanged as shown in Fig. 10. States of main specification function, which have the value of one, are presented in gray color in Fig. 10. Moreover, the values of states 0000 and 1000 have been changed to zero in Fig. 10(b). As the mentioned states in the main specification



**Fig. 11** a K-map used for function  $f_{3,2}$ . b K-map used for function  $f_{3,3}$ . Applying post-processing method to  $f_{3,2}$  and  $f_{3,3}$  for more simplification

function are zero, the number of ones placed in these cubes can be zero or one. In this example, Column 10 is common to two Rows 00 and 10, which have been exchanged with the same column in Fig. 10(a) ( $f_2$ ).

Then, states (1001, 1011) are fixed and thus, among of other states, states (0000, 0100) must be zero until it creates a square in K-map. Logic expressions for  $f_2$  and  $f_3$  are as follows:

$$f_2 = cd' = \text{Maj}(c, d', 0), \quad f_3 = b'd = \text{Maj}(b', d, 0). \tag{14}$$

The total logic function is

$$F = \text{Maj}(\text{Maj}(a, b, d), \text{Maj}(c, d', 0), \text{Maj}(b', d, 0)). \tag{15}$$

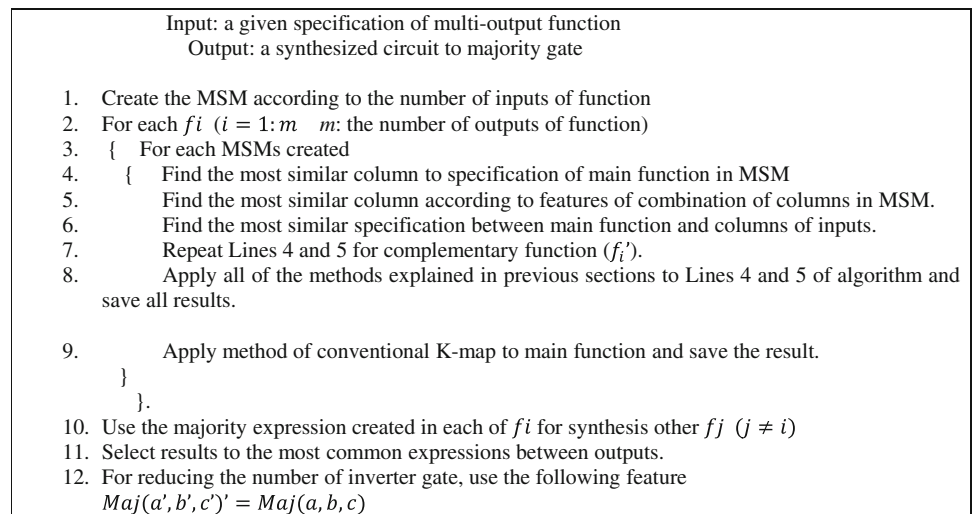
*Example 5* Consider the specification function  $F = (3, 4, 7, 15)$  which is defined as the first and the second columns of Table 13. The tree method is explained for this example.

First, the most similar column to main function is selected; here,  $\text{Maj}(0, c, d)$  is combined with  $\text{Maj}(b, c, d)$  and  $\text{Maj}(b', c, d)$  Then, the logic function  $f_2$  according to K-map shown in Table 14 is obtained ( $f_2 = b$ ).

For obtaining the specification function  $f_3$  as a tree method, its value is considered as the main function and is shown in Table 15.

As shown in Table 15, function  $f_3$  is considered as the main function. It is important to note that in the calculation of specification functions of  $F_{3,1}$ ,  $F_{3,2}$ , and  $F_{3,3}$ , the don't care values of the main function,  $F_3$ , are also considered to be don't cares, i.e., their values may be either zero or one.

**Fig. 12** The proposed flow for the synthesis of multi-output functions



**Table 16** Comparison between the proposed method with a multi-objective genetic programming method presented in [24]. NOI, NOM and NTG stand for number of inverter gates, number of majority gates and the total number of gates, respectively

| Function [24]                     | [24] |     |     | Proposed method |     |     | Synthesized circuit                                  |
|-----------------------------------|------|-----|-----|-----------------|-----|-----|--|
|                                   | NOI  | NOM | NTG | NOI             | NOM | NTG |  |
| $f_1 = (2, 4, 6)$                 | 3    | 4   | 7   | 1               | 2   | 3   | $M(M(a, b, c'), 0, c')$                              |
| $f_2 = (0, 1, 3, 6)$              | 3    | 4   | 7   | 3               | 4   | 7   | $M(M(a, M(a, b, c'), 1)', 1, M(a, b, 1'))$           |
| $f_3 = (0, 3, 6)$                 | 4    | 4   | 8   | 4               | 4   | 8   | $M(M(a', b, c'), M(M(a, b', 1), c, 1), 1')$          |
| Shared gates                      | 4    | 4   | 8   | 3               | 2   | 5   |  |
| Total number of gates             | 6    | 8   | 14  | 5               | 8   | 13  |  |
| $f_1 = (1, 4, 5, 7)$              | 1    | 1   | 2   | 1               | 1   | 2   | $M(a, b', c)$  |
| $f_2 = (3, 4, 6)$                 | 3    | 4   | 7   | 3               | 4   | 7   | $M(M(a, b, c), M(a', c', 1), M(a, c', 1'))$          |
| $f_3 = (0, 2, 5, 6)$              | 3    | 4   | 7   | 4               | 4   | 8   | $M(M(a, b', c'), M(a, c', 1), M(c, b', 1'))$         |
| $f_4 = (4, 6, 7)$                 | 3    | 3   | 6   | 2               | 2   | 4   | $M(M(a, c', 1'), a, b)$                              |
| Shared gates                      | 4    | 3   | 7   | 5               | 2   | 7   |  |
| Total number of gates             | 6    | 9   | 15  | 5               | 9   | 14  |  |
| $f_1 = (0, 3, 6, 7, 15)$          | 3    | 5   | 8   | 4               | 5   | 9   | $M(M(a', b, M(c', d, 1)), M(c, M(b, d, 1)', 1), 1')$ |
| $f_2 = (9, 11, 14)$               | 4    | 4   | 8   | 4               | 4   | 6   | $M(M(a, b, d), 1', M(a', b, M(c', d, 1))')$          |
| $f_3 = (8, 10, 11, 14, 15)$       | 1    | 3   | 4   | 2               | 3   | 5   | $M(M(M(b, d, 1)', c, 1), a, 1')$                     |
| Shared gates                      | 3    | 4   | 7   | 5               | 4   | 9   |  |
| Total number of gates             | 5    | 8   | 13  | 5               | 8   | 13  |  |
| $f_1 = (3, 4, 7, 15)$             | 3    | 4   | 7   | 2               | 4   | 6   | $M(M(1', c, d), b, M(M(a, b, c), b, d)')$            |
| $f_2 = (1, 3, 4, 9, 13, 15)$      | 4    | 5   | 9   | 3               | 5   | 8   | $M(M(M(a, b', 1), d, 1'), b, M(M(a, b, c), b, d)')$  |
| $f_3 = (3, 6, 7, 11, 13, 14, 15)$ | 2    | 3   | 5   | 2               | 3   | 5   | $M(M(M(a, b', 1), d, 1'), b, c)$                     |
| $f_4 = (2, 6, 10, 11, 14)$        | 5    | 5   | 10  | 3               | 5   | 8   | $M(M(M(1', c, d), b, M(M(a, b, c), b, d)')', 1, c)$  |
| Shared gates                      | 6    | 8   | 14  | 6               | 8   | 14  |  |
| Total number of gates             | 8    | 9   | 17  | 4               | 9   | 13  |  |

First, the most similar column to the main function is selected ( $M(a', b', c')$ ), and the logic functions of  $f_{3\_2}$  and  $f_{3\_3}$  are as follows:

$$f_{3\_2} = 0, \quad f_{3\_3} = d' + c = Maj(d'c, 1). \tag{16}$$

Then, specification functions of  $f_{3\_2}$  and  $f_{3\_3}$  are obtained using the post-processing method as shown in Fig. 11.

**Table 17** Comparisons of the proposed method with the methods presented in [28] and [29]

| Benchmarks         | Method [28] |                | Method [29] |                |           | Proposed method |                |          | Reduction% VS. [28] |                | Reduction% VS. [29] |                |                |
|--------------------|-------------|----------------|-------------|----------------|-----------|-----------------|----------------|----------|---------------------|----------------|---------------------|----------------|----------------|
|                    | Level       | Majority gates | Level       | Majority gates | Inverters | Level           | Majority gates | Inverter | Level               | Majority gates | Level               | Majority gates | Inverter gates |
| b1                 | 3           | 9              | 2           | 7              | 5         | 2               | 6              | 4        | 33.3                | 33.3           | 0.0                 | 14.3           | 20.0           |
| cm82a              | 4           | 16             | 3           | 7              | 6         | 3               | 6              | 4        | 25.0                | 62.5           | 0.0                 | 14.3           | 33.3           |
| majority           | 4           | 6              | 4           | 6              | 0         | 4               | 5              | 0        | 0.0                 | 16.6           | 0.0                 | 16.6           | 0.0            |
| 9symml             | 12          | 216            | 10          | 47             | 23        | 10              | 47             | 18       | 16.6                | 78.2           | 0.0                 | 0.0            | 21.7           |
| x2                 | 7           | 42             | 7           | 37             | 15        | 6               | 34             | 11       | 14.2                | 19.0           | 14.3                | 8.1            | 26.6           |
| cm152a             | 6           | 21             | 6           | 21             | 7         | 4               | 15             | 3        | 33.3                | 28.5           | 33.3                | 28.5           | 57.1           |
| cm85a              | 7           | 34             | 6           | 26             | 12        | 6               | 14             | 9        | 14.3                | 58.8           | 0.0                 | 46.1           | 25.0           |
| cm151a             | 7           | 42             | 7           | 23             | 10        | 4               | 15             | 5        | 42.8                | 64.2           | 42.8                | 34.7           | 50.0           |
| cm162a             | 7           | 46             | 7           | 41             | 14        | 8               | 32             | 11       | -14.3               | 30.4           | -14.3               | 21.9           | 21.4           |
| cu                 | 7           | 46             | 7           | 40             | 21        | 5               | 36             | 12       | 28.5                | 21.7           | 28.5                | 10.0           | 42.8           |
| cm163a             | 7           | 42             | 7           | 38             | 17        | 6               | 28             | 16       | 14.3                | 33.3           | 14.3                | 26.3           | 5.0            |
| cmb                | 4           | 44             | 4           | 28             | 4         | 4               | 26             | 2        | 0.0                 | 40.9           | 0.0                 | 7.0            | 50.0           |
| pm1                | 6           | 45             | 6           | 35             | 16        | 6               | 30             | 13       | 0.0                 | 33.3           | 0.0                 | 14.3           | 18.7           |
| cm150a             | 9           | 46             | 9           | 46             | 20        | 6               | 37             | 10       | 33.3                | 19.5           | 33.3                | 19.5           | 50.0           |
| mux                | 9           | 46             | 9           | 46             | 12        | 5               | 35             | 4        | 44.4                | 23.9           | 44.4                | 23.9           | 66.6           |
| il                 | 6           | 41             | 6           | 36             | 12        | 6               | 32             | 4        | 0.0                 | 21.9           | 0.0                 | 11.1           | 66.6           |
| decod              | 3           | 28             | 3           | 28             | 6         | 3               | 28             | 4        | 0.0                 | 0.0            | 0.0                 | 0.0            | 33.3           |
| pcl                | 8           | 67             | 8           | 62             | 17        | 7               | 48             | 18       | 12.5                | 28.3           | 12.5                | 22.6           | -5.0           |
| tcon               | 2           | 24             | 2           | 24             | 8         | 2               | 24             | 1        | 0.0                 | 0.0            | 0.0                 | 0.0            | 87.5           |
| cc                 | 5           | 44             | 5           | 43             | 8         | 5               | 36             | 8        | 0.0                 | 18.1           | 0.0                 | 16.3           | 0.0            |
| Average reductions |             |                |             |                |           |                 |                |          | 14.9                | 31.6           | 10.5                | 16.8           | 33.5           |

New Boolean logic functions  $f_{3\_2}$  and  $f_{3\_3}$  are as follows:

$$f_{3\_2} = b', \quad f_{3\_3} = d'. \quad (17)$$

The total specification function is

$$F = \text{Maj}(\text{Maj}(0, c, d), b, \text{Maj}(M(a', b', c'), b', d')). \quad (18)$$

#### 4 A synthesis flow for multi-output functions

Having the proposed approaches from the previous section available, they can be combined with an extended synthesis flow that could be used for multi-output functions. Figure 12 illustrates this flow. As shown in this figure, first, according to the numbers of inputs of the main function, MSMs are created; then, for each of the outputs of function ( $f_i$ ) and their complementaries, the most similar column of MSM or inputs of function is selected, as well as it could be selected from the combination of columns in MSM that led to AND/OR functions. Then, for each of the selected columns, the proposed methods in previous sections are applied. Afterwards, the obtained results are saved. In addition, conventional K-map method is applied separately and its result is saved. Because

of that, in some functions, expression obtained from this method is simpler. In order to obtain final terms including the most common expressions, in Line 10 of the algorithm, the output expressions of the previous functions are used in the synthesis of other functions. Then the results based on the objectives of priority gate counts and gate levels are ordered, and after that results to the most common terms are selected. Finally, for reducing the number of inverter gates in the final expressions, Line 12 of algorithm is applied.

#### 5 Results and comparison

In this section, first, a comparison between the proposed synthesis flow and a multi-objective genetic programming approach [24] is made. The results are shown in Table 16. In this table, NOI, NOM, and NTG stand for the number of inverter gates, number of majority gates, and the total number of gates, respectively. The obtained synthesis results are shown in the last column of this table. In this column, the common parts of the corresponding synthesized circuit for each output are underlined. Also, in this table, the rows titled “shared gates” and “total number of gates” show the number of shared gates and the total number of gates used in the

function outputs, respectively. The results illustrate that the proposed method produces the same or better results than the approach presented in [24].

In some functions, the two parameters, namely the number of gates and the number of common parts, should be considered simultaneously, e.g., for the specification function  $F = (2, 6, 10, 11, 14)$ , the following expressions are obtained (this function is a sample of a multi-output function in Table 16):

$$\begin{aligned} F_1 &= \text{Maj}(\text{Maj}(\text{Maj}(a, b', 0), d', 1), c, 0) \\ F_2 &= \text{Maj}(\text{Maj}(\text{Maj}(1', c, d), b, \text{Maj}(\text{Maj}(a, b, c), b, d)'), 1, c). \end{aligned} \quad (19)$$

Although  $F_1$  produces a fewer number of gates,  $F_2$  is chosen because it leads to the large number of common parts with the other outputs presented in Table 16. The common parts lead to a high reduction in the total number of gates.

In Table 17, an overall comparison between the best existing majority logic synthesis method [29] and the proposed synthesis flow is demonstrated. Also, in this table the proposed synthesis flow is compared with the method presented in [28]. In this table, the first column lists the names of benchmarks. The columns titled “Method [28]” and “Method [29]” show the results for the corresponding benchmarks obtained from [28] and [29] in terms of the number of levels, gates, and the number of inverter gates only for [29], as the method in [28] has not reported the number of inverter gates. The column titled “Proposed method” shows the results obtained from the proposed synthesis flow in terms of the number of levels, gates, and the number of inverter gates. The “Reduction %” columns compare the proposed method with the methods presented in [28] and [29] and give the percent-

age reductions. This table illustrates that there is an average reduction of 14.9% in the number of levels and, at the same time, the number of gates is reduced by 31.6% as compared to the method presented in [28]. When compared to [29], the average reduction in levels is 10.5%, the reduction in gate counts is 16.8%, and the reduction in inverter gates is 33.5%. Results show that the proposed synthesis flow outperforms the best existing methods. Also, the detailed results of the proposed synthesis flow applied to MCNC benchmarks are shown in Appendix (Table 18). Since the synthesis results are independent of the technology used, they are effective for any majority/minority-based technology including ASL, QCA, SET, and TPL.

## 6 Conclusion

In this paper, a multi-objective synthesis methodology for generating optimal majority expressions was presented. In this method, using an MSM and a synthesis flow presented for multi-output specification functions, the majority expressions are optimized. The corresponding minority network can be easily obtained by complementing the majority expressions. The proposed approach was applied to 20 MCNC benchmarks and was compared with the previous methods. The experimental results demonstrated that the proposed method outperforms the best existing ones.

## Appendix

In this section, the circuits obtained by applying the proposed method to 20 MCNC benchmarks are illustrated in Table 18.

**Table 18** Circuits obtained by applying the proposed method to 20 MCNC benchmarks

| Benchmarks | Circuit  |
|------------|--|
| b1         | $f = M(M(a, b', 1)), M(a', c, 0), M(b, c', 0)), e = M(M(a, b', 0), \underline{M(a, b', 1)'}, 1), d = c, g = c'$  |
| cm82a      | $f = M(M(a, b, c)', M(a, b', c), b), g = M(M(M(a, b, c), d, e)', \underline{M(M(a, b, c), d', e)}, d), h = \underline{M(M(a, b, c), d, e)}$  |
| majority   | $f = M(d, M(M(a, b, M(c, e, 0)), M(e, c, 1), 0), 1)$   |
| 9symml     | $(52) = M(M(M([7815], M([7819]', M(M([7816]', [7817], (0)), M([7818], [7820]', (0)), (1)), M([7816], [7818]', (0))),$<br>$M([7817]', M([7816]', [7820], (0)), M([7819], [7820]', (0))), M([7815]', M(M([7816], M([7817], [7819]', (0))),$<br>$\underline{M([7818]', [7819], (0))), M([7817], M([7816]', [7820], (0)), M([7818], [7820]', (0))), (1)),$<br>$M(M([7817]', [7818], (0)), [7819], (0)), (1)), M(M(M([7817]', [7820], (0)), M(M([7816], [7818], (0)), [7819]', (0)),$<br>$\underline{M([7818]', [7819], (0))), M(M(M([7816]', [7817], (0)), \underline{M([7819], [7820]', (0)), (0)), [7818]', (0)), (1)), (1)),$<br>$[7815] = M(1, 2, 3),$<br>$[7816] = M(M(1, \underline{M(2, 3, (0))}, M(2', 3', (0))), M(1', M(M(2, 3, (0)), \underline{M(2', 3', (0))}, 1)', 0), 1),$<br>$[7817] = M(4, 5, 6),$<br>$[7818] = M(M(4, \underline{M(5, 6, (0))}, M(5', 6', (0))), M(4', M(M(5, 6, (0)), \underline{M(5', 6', (0))}, 1)', 0), 1),$<br>$[7819] = M(7, 8, 9),$<br>$[7820] = M(M(7, \underline{M(8, 9, (0))}, M(8', 9', (0))), M(7', M(M(8, 9, (0)), \underline{M(8', 9', (0))}, 1)', 0), 1)$ |
| x2         | $\langle k \rangle = M(i', 1, M(h', j, 1)), \langle L \rangle = M((M(h, j, 0), \underline{M(h, j, 1)'}, i')', \langle m \rangle = M(M(h', i', 0), j', 0),$<br>$\langle n \rangle = M(M(i', \underline{M(h, j, 1)}, 1), \underline{M(M(a', b', 0), c', 0)'}, 1), \langle o \rangle = M(M(\underline{M(i, j, 0), h', 1}), \underline{g'}, 1),$<br>$\langle p \rangle = M(M(M(M(M(a', b', 0), c, 0), \underline{M(M(i, j, 0), h, 0)}, M(h', i', 0)),$<br>$M(j', 0, M(i', 1, M(M(d, e', 0), h, 0))), 1), \underline{M(g', 1, M(M(f, h', 0), M(i, j, 0), 0))}, 1),$<br>$\langle q \rangle = M(M(M(M(h, j, 0), \underline{M(M(a', b', 0), c', 0)'}, i', 1), 0), M(j', \underline{M(h', i', 0)}, M(M(d, e, 0), M(h, i, 0), 0)), 1),$<br>$\underline{M(g', 1, M(M(f, h', 0), M(i, j, 0), 0))}, 1)$   |
| cm152a     | $M(M(i', M(k', M(a, j', 0), M(c, j, 0))), M(k, M(e, j', 0), M(g, j, 0))), M(i, M(k', M(b, j'$<br>$, 0), M(d, j, 0)), M(k, M(f, j', 0), M(h, j, 0))), 1)$   |
| cm85a      | $L = M(M(M(M(M(h', i, M(j', k, 0)), f', g), d', e), b, 0), a, 1),$<br>$n = M(M(b, 0, M(M(M(h, i', M(j, k', 0)), f, g'), d, e')), c, 1),$<br>$m = M(b, 0, M(M(M(M(M(h', i, M(j', k, 0)), f', g), d', e), M(M(M(h, i', M(j, k', 0)), f, g'), d, e'), 1)')$   |
| cm151a     | $\langle m \rangle = M(L', M(j, M(k, M(g, i', 0), M(h, i, 0))), M(k', M(c, i', 0), M(d, i, 0))), M(j'$<br>$, M(k, M(e, i', 0), M(f, i, 0)), M(k', M(a, i', 0), M(b, i, 0))), \langle n \rangle =$<br>$\langle m \rangle'$  |
| cm162a     | $\langle o \rangle = M([556], f', 1), \langle p \rangle = M(\underline{M([559], f', 1)}, [593], 1), \langle q \rangle = M(\underline{M([559], f', 1)}, [627], 1),$<br>$\langle r \rangle = M(\underline{M([559], f', 1)}, [663], 1), \langle s \rangle = M(M(e, j, 0), n, 0), [45] = M(\underline{M(c, e, 0)}, i', 0),$<br>$[502] = \underline{M([45]', k, 1)}, [554] = M(\langle s \rangle, \underline{M(c, e, 0)'}, 1), [555] = M([45], M([554], i, 0), 1),$<br>$[556] = M([555], d, 0), M(a, d, 1)', 1), [559] = M(\langle s \rangle, M(c, d, 0), 0),$<br>$[592] = M(\underline{M([45]', k, 1)'}, M([45]', k, 0), 1), [593] = M(M([592], d, 0), M(b, d, 1)', 1),$<br>$[627] = M(M(M([502], L, 0), \underline{M([502], L, 1)'}, d), M(d, g, 1)', 1),$<br>$[663] = M(M(M(m, M([502], L, 1), 0), M(M([502], L, 1), m, 1)', d), M(d, h, 1)', 1)$  |
| cu         | $\langle p \rangle = M(M(M(c, e', 0), M(c', f', 0), 1), M(M(e, f, 0), d, 1), 1),$<br>$\langle q \rangle = M([1402], M([1404], M(c, f', 0), 0), 1), \langle r \rangle = M(\underline{M(\langle w \rangle, a', 0)}, b', 0),$<br>$\langle s \rangle = M(\underline{M(\langle w \rangle, a, 0)}, b', 0), \langle t \rangle = M(M(\langle w \rangle, a', 0), b, 0), \langle u \rangle = M(\underline{M(\langle w \rangle, a, 0)}, b, 0),$<br>$\langle v \rangle = M([1404], [1398], M(M([1403], i', 0), M(a, M(b, m', 0), M(b, k, 1)')), M(a', M(b, L', 0), M(b', j, 0))), \langle w \rangle$<br>$= M([1402], o', 0), \langle x \rangle = M([1404], [1398], [1403]), \langle y \rangle = M(g, o, 0),$   |

**Table 18** continued

| Benchmarks | Circuit  |
|------------|--|
| cm163a     | $\langle z \rangle = M(M(d', g, 0), M(c', f', 1), 0), [1398] = M(M(c, f', 0), o', 0),$<br>$[1402] = M(M(c', f, 0), M(d', e', 0), 0), [1403] = M(M(c', f, 0), M(n', o, 0), 0), [1404] = M(d', e, 0)$<br>$\langle q \rangle = M(f, M(a, e', 0), M(e, M(j, L0', 0), M(j', L0, 0)))',$<br>$\langle r \rangle = M(f, M(b, e', 0), M(e, M(L, m0, 0), M(L, m0, 1)'))',$<br>$\langle s \rangle = M(f, M(g, e', 0), M(e, M(m, q0, 0), M(m, q0, 1)'))',$<br>$\langle t \rangle = M(f, M(e', h, 0), M(e, M(n, r0, 0), M(n, r0, 1)'))', \langle u \rangle = M(d, p0', 0),$<br>$p0 = M(M(i, k, 0), M(o, p, 0), 0)', r0 = M(M(j, L0, 1), M(L, m, 1), 1)', q0 = M(M(j, L0, 1), L, 1)',$<br>$m0 = M(j, L0, 1)', L0 = M(c, d, 0)'$  |
| cmb        | $\langle q \rangle = M(M(M(M(a, b, 0), M(c, d, 0), 0), M(M(e, f, 0), M(g, h, 0), 0), 0), M(M(i, j, 0), M(k, L, 0), 0), 0),$<br>$\langle r \rangle = M(M(M(M(e, f, 1), M(g, h, 1), 1), M(M(i, j, 1), M(k, L, 1), 1), 1), M(M(m, n, 1), M(o, p, 1), 1), 1),$<br>$\langle s \rangle = M(M(M(M(e, f, 0), M(g, h, 0), 0), M(M(i, j, 0), M(k, L, 1), 0), 0), M(M(m, n, 0), M(o, p, 0), 0), 0)',$<br>$\langle t \rangle = M(M(M(M(e, f, 1), M(g, h, 1), 1), M(M(i, j, 1), M(k, L, 1), 1), 1), M(M(m, n, 1), M(o, p, 1), 1), 1)'$  |
| pm1        | $\langle r \rangle = M(b, M(m, n, 1), 1), \langle s \rangle = M(m', n, 0)',$<br>$\langle t \rangle = M(M(M(k, n, 0), m, 0), M(M(g, h, 0), M(i, j, 0), 0), 0)',$<br>$\langle u \rangle = M(M(M(k, n, 0), m, 0)', M(M(g, h, 0), M(i, j, 0), 0), 1), \langle v \rangle = p', \langle w \rangle = o',$<br>$\langle x \rangle = M(M(b, k, 0), M([2897], M(m', n, 0), 1), 0),$<br>$\langle z \rangle = M(M(a', M(M(M(c, d, 0), e, 0), M(k, n, 0), 0), 1), M(L, m', 1), 1),$<br>$\langle a0 \rangle = M(M(a, L', 0), M(m, n, 0), M(m, n, 1)') \langle b0 \rangle = M(M([2897], [2901], 0), M(k, n, 0), 0)$<br>$\langle c0 \rangle = M(M(\langle b0 \rangle, b, 0), M([2901], M(b, n, 1)', 0), 1), \langle d0 \rangle = M([2901], M(k', n, 0), 0),$<br>$[2897] = M(M(c, d, 0), e, 0)', [2898] = M(M(k, n, 0), m, 0)', [2901] = M(M(a, L', 0), m, 0)$   |
| cm150a     | $\langle v \rangle = M(u', f1, f2)',$<br>$f1 = M(t', M(s', M(r', M(a, q', 0), M(b, q, 0)), M(M(M(c', q', 0), M(d', q, 0), 1)'$<br>$, r, 0)), M(s, M(r', M(e, q', 0), M(f, q, 0)), M(r, 0, M(M(g', p', 0), M(h', q, 0), 1)'))),$<br>$f2 = M(t, 0, M(M(M(r', M(i, q', 0), M(j, q, 0)), M(r, 0, M(M(k', q', 0), M(L', q, 0), 1)')), 1),$<br>$M(M(r', M(m, q', 0), M(n, q, 0)), M(r, 0, M(M(o', q', 0), M(p', q, 0), 1)')), s),$<br>$M(s', 1, M(M(r', M(m, q', 0), M(n, q, 0)), M(r, 0, M(M(o', q', 0), M(p', q, 0), 1)')), s))'$  |
| mux        | $M(u, M(M(q, M(M(s, t, 0), M(a, r, 0), M(e, r', 0)), M(M(s, t', 0), M(b, r, 0), M(f, r', 0))), M(q, M(M(s', t, 0),$<br>$M(c, r, 0), M(g, r', 0)), M(M(s', t', 0), M(d, r, 0), M(h, r', 0))), 1), M(M(q', M(M(s, t, 0), M(i, r, 0), M(m, r', 0)), M$<br>$(M(s, t', 0), M(j, r, 0), M(n, r', 0))), M(q', M(M(s', t, 0), M(k, r, 0), M(o, r', 0)), M(M(s', t', 0), M(l, r, 0), M(p, r', 0))), 1)$   |
| il         | $\langle V27\_1 \rangle = M(M((in\_V27\_0)', (in\_V29\_0), 0), M([33], M(V8\_0, V9\_0, 0), M((V8\_0)', (V9\_0)', 0))), 1),$<br>$\langle V27\_2 \rangle = M(M(M((in\_V27\_0), (in\_V29\_0), 0), M(M(M(V7\_1, V7\_2, 1), M(V7\_3, V7\_4, 1), 1),$<br>$M(M(V7\_5, V7\_6, 1), V7\_7, 1), 1), 0), M(M(V8\_0, (V9\_0)', 0), [33], 0), 1),$<br>$\langle V27\_4 \rangle = M(in\_V27\_3, V22\_2, 1), \langle V28\_0 \rangle = M(V10\_0, M(V8\_0', [33], 0), 1),$<br>$\langle V30\_0 \rangle = M(V18\_0, V22\_5, 0), \langle V32\_0 \rangle = M(V11\_0, V22\_5, 0),$<br>$\langle V33\_0 \rangle = M(V14\_0, M(V22\_3, (V22\_5)', 0), 0),$<br>$\langle V34\_0 \rangle = M(V17\_0, M(V22\_3, (V22\_5)', 0), 0),$<br>$\langle V35\_0 \rangle = M(V14\_0, M(V22\_4, (V22\_5)', 0), 0),$<br>$\langle V36\_0 \rangle = M(V17\_0, M(V22\_4, (V22\_5)', 0), 0), \langle V37\_0 \rangle = M(V16\_0, (V22\_5)', 0),$<br>$\langle V38\_0 \rangle = M(M(V12\_0, V13\_0, 1), M(V14\_0, V15\_0, 1), 1),$<br>$\langle V27\_0 \rangle = in\_V27\_0, \langle V27\_3 \rangle = in\_V27\_3, \langle V29\_0 \rangle = in\_V29\_0, \langle V31\_0 \rangle = V11\_0,$<br>$[33] = M(in\_V29\_0, M(M(M(V7\_1, V7\_2, 1), M(V7\_3, V7\_4, 1), 1), M(M(V7\_5, V7\_6, 1), V7\_7, 1), 1)', 0)$ |



Table 18 continued

| Benchmarks | Circuit   |
|------------|---|
| decod      | $f = M(\underline{M(a, d, 0)}, \underline{M(e, M(b, c, 0), 0)}, 0), g = M(\underline{M(a, d', 0)}, \underline{M(e, M(b, c, 0), 0)}, 0),$<br>$h = M(\underline{M(a, d, 0)}, \underline{M(e, M(b, c', 0), 0)}, 0), i = M(\underline{M(a, d', 0)}, \underline{M(e, M(b, c', 0), 0)}, 0),$<br>$j = M(\underline{M(a, d, 0)}, \underline{M(e, M(b', c, 0), 0)}, 0), k = M(\underline{M(a, d', 0)}, \underline{M(e, M(b', c, 0), 0)}, 0),$<br>$L = M(\underline{M(a, d, 0)}, \underline{M(e, M(b', c', 0), 0)}, 0), M = M(\underline{M(a, d', 0)}, \underline{M(e, M(b', c', 0), 0)}, 0),$<br>$N = M(\underline{M(a', d, 0)}, \underline{M(e, M(b, c, 0), 0)}, 0), O = M(\underline{M(a', d', 0)}, \underline{M(e, M(b, c, 0), 0)}, 0),$<br>$P = M(\underline{M(a', d, 0)}, \underline{M(e, M(b, c', 0), 0)}, 0), Q = M(\underline{M(a', d', 0)}, \underline{M(e, M(b, c', 0), 0)}, 0),$<br>$R = M(\underline{M(a', d, 0)}, \underline{M(e, M(b', c, 0), 0)}, 0), S = M(\underline{M(a', d', 0)}, \underline{M(e, M(b', c, 0), 0)}, 0),$<br>$T = M(\underline{M(a', d, 0)}, \underline{M(e, M(b', c', 0), 0)}, 0), U = M(\underline{M(a', d', 0)}, \underline{M(e, M(b', c', 0), 0)}, 0)$ |
| pcle       | $\langle t \rangle = M(M(S, V0, 0), U0, 0), \langle u \rangle = M(M(a, i, 0), M(L', U0, 0), 1),$<br>$\langle v \rangle = M(M(b, i, 0), M(M(L, m', 0), M(L', m, 0), U0), 1),$<br>$\langle w \rangle = M(M(c, i, 0), M(M(a1, n', 0), M(a1', n, 0), U0), 1),$<br>$\langle x \rangle = M(M(d, i, 0), M(M(o, Z0', 0), M(o', Z0, 0), U0), 1),$<br>$\langle y \rangle = M(M(e, i, 0), M(U0, M(M(p, M(o, Z0, 0), 0), M(p', M(o, Z0, 0)', 0), 1)', 0), 1),$<br>$\langle z \rangle = M(M(f, i, 0), M(M(q, x0', 0), M(q', x0, 0), U0), 1),$<br>$\langle a0 \rangle = M(M(g, i, 0), M(U0, M(M(r, M(q, x0, 0), 0), M(r', M(q, x0, 0)', 0), 1)', 0), 1),$<br>$\langle b0 \rangle = M(M(h, i, 0), M(U0, M(S, v0', 0), M(s', v0, 0), 1), U0 = M(M(i', j, 0), k', 0),$<br>$V0 = M(M(r, q, 0), x0, 0), a1 = M(L, m, 0), Z0 = M(a1, n, 0), x0 = M(M(o, p, 0), Z0, 0),$   |
| tcon       | $\langle a0 \rangle = M(a, M(i, k, 1), M(i', k, 0)), \langle b0 \rangle = M(b, M(i, L, 1), M(i', L, 0)),$<br>$\langle c0 \rangle = M(c, M(i, m, 1), M(i', m, 0)), \langle d0 \rangle = M(d, M(i, n, 1), M(i', n, 0)),$<br>$\langle e0 \rangle = M(e, M(i, o, 1), M(i', o, 0)), \langle f0 \rangle = M(f, M(i, p, 1), M(i', p, 0)),$<br>$\langle g0 \rangle = M(g, M(i, q, 1), M(i', q, 0)), \langle h0 \rangle = M(h, M(i, r, 1), M(i', r, 0)), \langle s \rangle = k, \langle t \rangle = L, \langle u \rangle = m,$<br>$\langle v \rangle = n, \langle w \rangle = o, \langle x \rangle = p, \langle y \rangle = q, \langle z \rangle = r$  |
| cc         | $\langle w \rangle = M(L, v, 0), \langle x \rangle = M(M(M(i, k, 0), q, 0), p', 0),$<br>$\langle y \rangle = M(M(L', m, 0), M(M(M(i, k, 0), q', 0), p, 1), 0), \langle z \rangle = M(\langle x \rangle, m, 0), \langle a0 \rangle = r', \langle f0 \rangle = M(i, j, 0),$<br>$\langle g0 \rangle = M(i, j, 0)', \langle i0 \rangle = M(m, M([84], o, 0), M(a, M(M(i, k, 0), q', 0), 0)),$<br>$\langle j0 \rangle = M(m, M(M(p, M(b, M(i, k, 0), 0)', 0), M(M(i, k, 0), q, 0), 1), 0),$<br>$\langle k0 \rangle = M(m, M(c, M(M(i, k, 0), q', 0), 0), M(q, M(i, k, 0)', 0)),$<br>$\langle L0 \rangle = M(m, M(M([84], p', 0), r, 0), M(d, 0, M(M(i, k, 0), q', 0))),$<br>$\langle m0 \rangle = M(m, M([84], s, 0), M(e, 0, M(M(i, k, 0), q', 0))),$<br>$\langle n0 \rangle = M(m, M([84], t, 0), M(f, 0, M(M(i, k, 0), q', 0))),$<br>$\langle o0 \rangle = M(m, M([84], u, 0), M(g, 0, M(M(i, k, 0), q', 0))),$<br>$\langle p0 \rangle = M(m, M([84], v, 0), M(h, 0, M(M(i, k, 0), q', 0))), \langle b0 \rangle = u, \langle c0 \rangle = q, \langle d0 \rangle = s, \langle e0 \rangle = r,$<br>$\langle h0 \rangle = p, [84] = M(M(i, k, 0), q', 0)'$                               |

## References

- International Technology Roadmap for Semiconductors (ITRS) (2013). <http://www.itrs.net>
- Snider, G., Orlov, A., Amlani, I., Zuo, X., Bernstein, G., Lent, C., Merz, J., Porod, W.: Quantum-dot cellular automata: review and recent experiments. *J. Appl. Phys.* **85**(8), 4283–4285 (1999)
- Oya, T., Asai, T., Fukui, T., Amemiya, Y.: A majority-logic nanodevice using a balanced pair of single-electron boxes. *J. Nanosci. Nanotechnol.* **2**(3–4), 333–342 (2002)
- Oya, T., Asai, T., Fukui, T., Amemiya, Y.: A majority-logic device using an irreversible single-electron box. *IEEE Trans. Nanotechnol.* **2**(1), 15–22 (2003)
- Fahmy, H., Kiehl, R.A.: Complete logic family using tunneling phase-logic devices. In: *Proceedings of the International Conference on Microelectronics*, pp. 22–24 (1999)
- Augustine, C., Panagopoulos, G., Behin-Aein, B., Srinivasan, S., Sarkar, A., Roy, K.: Low-power functionality enhanced computation architecture using spin-based devices. In: *2011 IEEE/ACM International Symposium on Nanoscale Architectures*, pp. 129–136 (2011)
- Zhu, J.-G.J., Park, C.: Magnetic tunnel junctions. *Mater. Today* **9**(11), 36–45 (2006)
- Datta, S., Das, B.: Electronic analog of the electro-optic modulator. *Appl. Phys. Lett.* **56**(7), 665–667 (1990)
- Sugahara, S., Nitta, J.: Spin-transistor electronics: an overview and outlook. *Proc. IEEE* **98**(12), 2124–2154 (2010)

10. Lent, C.S., Tougaw, P.D., Porod, W.: Bistable saturation in coupled quantum dots for quantum cellular automata. *Appl. Phys. Lett.* **62**(7), 714–716 (1993)
11. Tougaw, P.D., Lent, C.S., Porod, W.: Bistable saturation in coupled quantum-dot cells. *J. Appl. Phys.* **74**(5), 3558–3566 (1993)
12. Bourianoff, G., Brillouet, M., Cavin III, R.K., Hiramoto, T., Hutchby, J.A., Ionescu, A.M., Uchida, K.: Nanoelectronics research for beyond CMOS information processing. *Proc. IEEE* **98**(12), 1986–1992 (2010)
13. Hutchby, J., Bourianoff, G., Zhirnov, V.V., Brewer, J.E.: Extending the road beyond CMOS. *IEEE Circ. Dev. Mag.* **18**(2), 28–41 (2002)
14. Behin-Aein, B., Datta, D., Salahuddin, S., Datta, S.: Proposal for an all-spin logic device with built-in memory. *Nat. Nanotechnol.* **5**(4), 266–270 (2010)
15. Miller, H., Winder, R.: Majority-logic synthesis by geometric methods. *IEEE Trans. Electron. Comput.* **1**(EC-11), 89–90 (1962)
16. Zhang, R., Walus, K., Wang, W., Jullien, G.: A method of majority logic reduction for quantum cellular automata. *IEEE Trans. Nanotechnol.* **3**(4), 443–450 (2004)
17. Karnaugh, M.: The map method for synthesis of combinational logic circuits. *Commun. Electron.* **72**(5), 593–599 (1953)
18. Akers Jr, S.B.: Synthesis of combinational logic using three-input majority gates. In: *Proceedings of the Third Annual Symposium on Switching Circuit Theory and Logical Design, SWCT*, pp. 149–158 (1962)
19. Walus, K., Schulhof, G., Jullien, G., Zhang, R., Wang, W.: Circuit design based on majority gates for applications with quantum-dot cellular automata. In: *Conference Record of the Thirty-Eighth Asilomar Conference on Signals, Systems and Computers*, pp. 1354–1357 (2004)
20. Huo, Z., Zhang, Q., Haruehanroengra, S., Wang, W.: Logic optimization for majority gate-based nanoelectronic circuits. In: *Proceedings. 2006 IEEE International Symposium on Circuits and Systems, ISCAS*, pp. 4–1310 (2006)
21. Bonyadi, M., Azghadi, S., Rad, N., Navi, K., Afjei, E.: Logic optimization for majority gate-based nanoelectronic circuits based on genetic algorithm. In: *International Conference on Electrical Engineering, ICEE'07*, pp. 1–5 (2007)
22. Houshmand, M., Khayat, S.H., Rezaei, R.: Genetic algorithm based logic optimization for multi-output majority gate-based nanoelectronic circuits. In: *IEEE International Conference on Intelligent Computing and Intelligent Systems, ICIS*, pp. 584–588 (2009)
23. Houshmand, M., Saleh, R.R., Houshmand, M.: Logic minimization of QCA circuits using genetic algorithms. In: *Soft Computing in Industrial Applications*, pp. 393–403. Springer, Berlin (2011)
24. Rezaee, R., Houshmand, M., Houshmand, M.: Multi-objective optimization of QCA circuits with multiple outputs using genetic programming. *Genet. Program. Evolvable Mach.* **14**(1), 95–118 (2012). doi:[10.1007/s10710-012-9173-6](https://doi.org/10.1007/s10710-012-9173-6)
25. Wang, P., Niamat, M., Vemuru, S.: Minimal majority gate mapping of 4-variable functions for quantum cellular automata. In: *11th IEEE Conference on Nanotechnology (IEEE-NANO)*, pp. 1307–1312 (2011)
26. Wang, P., Niamat, M., Vemuru, S.: Minimal majority gate mapping of four-variable functions for quantum-dot cellular automata. In: *Nanoelectronic Device Applications Handbook*, pp. 263–280
27. Wang, P., Niamat, M., Vemuru, S.: 4-variable standard function majority gate logic synthesis using graph isomorphism. In: *Field-Coupled Nanocomputing Workshop*, University of South Florida, Tampa (2013)
28. Zhang, R., Gupta, P., Jha, N.K.: Majority and minority network synthesis with application to QCA-, SET-, and TPL-based nanotechnologies. *IEEE Trans. Comput.-Aided Des. Integr. Circ. Syst.* **26**(7), 1233–1245 (2007)
29. Kong, K., Shang, Y., Lu, R.: An optimized majority logic synthesis methodology for quantum-dot cellular automata. *IEEE Trans. Nanotechnol.* **9**(2), 170–183 (2010)
30. Amlani, I., Orlov, A.O., Kummmamuru, R.K., Bernstein, G.H., Lent, C.S., Snider, G.L.: Experimental demonstration of a leadless quantum-dot cellular automata cell. *Appl. Phys. Lett.* **77**(5), 738–740 (2000)
31. Augustine, C., Fong, X., Behin-Aein, B., Roy, K.: Ultra-low power nanomagnet-based computing: a system-level perspective. *IEEE Trans. Nanotechnol.* **10**(4), 778–788 (2011)
32. Yu, Z., Dutton, R.W., Kiehl, R.A.: Circuit/device modeling at the quantum level. *IEEE Trans. Electron Dev.* **47**(10), 1819–1825 (2000)
33. Shedsale, R., Sarwade, N.: A review of construction methods for regular LDPC codes. *Indian J. Comput. Sci. Eng. (IJCSSE)* **3**, 380–385 (2012)
34. Ammar, B., Honary, B., Kou, Y., Xu, J., Lin, S.: Construction of low-density parity-check codes based on balanced incomplete block designs. *IEEE Trans. Inf. Theory* **50**(6), 1257–1268 (2004)
35. Honary, B., Lin, S., Gabidulin, E.M., Xu, J., Kou, Y., Moinian, A., Ammar, B.: On construction of low density parity check codes. Paper presented at the 2nd International Workshop: Signal Processing for Wireless Communications 2004 (SPWC 2004), London (2004)