A multi-objective synthesis methodology in quantum-dot cellular automata technology

Moein Sarvaghad-Moghaddam¹, Ali A. Orouji¹ and Monireh Houshmand²

¹ Department of Electrical and Computer Engineering, Semnan University, Semnan, Iran
² Department of Electrical and Computer Engineering, Imam Reza International University, Mashhad, Iran

Email: aliaorouji@semnan.ac.ir

Abstract

Quantum-dot Cellular Automata (QCA) has been widely advocated in nanotechnology as a response to the physical limits associated with complementary metal oxide semiconductor (CMOS) technology in atomic scales. Some of its peculiar features are its smaller size, higher speed, higher switching frequency, lower power consumption, and higher scale integration. In this technology, the majority and NOT gates are employed for the production of the functions as these two gates together make a universal set of Boolean primitives in QCA technology. 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 NOT 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 Programing (GP). Besides, it is applied to synthesize MCNC benchmark circuits. The results are indicative of the outperformance of the proposed method in comparison to 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 NOT gates, as compared to the best available method respectively.

Index Terms: 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) [2-5], Single-Electron Tunneling (SET) [6, 7], and Tunneling Phase Logic (TPL) [8] can be used as possible alternatives to CMOS.

QCA was first proposed by Lent (1993) [9, 10]. QCA is a promising transistor-less technology and beyond-CMOS technology and will play a crucial role in the future of supercomputing [11, 12]. 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 NOT gates are the fundamental primitives.

In CMOS technology, “NAND/NOR/NOT” gates are used to implement circuits; thus, methods created for 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 (POS), are not efficient enough for synthesis of functions to present the simplest possible form for the QCA technology.

Some of researchers [13-16] 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 [14, 17] are based on a geometrical interpretation of only three-variable Boolean functions to reduce the majority expressions created by sum of products. These methods
The QCA devices are the three inputs. 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 [19-22] have applied meta-heuristic algorithms such as Genetic Algorithm (GA) and Genetic Programming (GP) for simplification of logic functions. Bonyadi et al. [19] used GA for optimization of a given single-output Boolean function by majority and NOT gates while Houshmand et al. [20, 21] applied GP algorithm for optimization of multi-outputs functions. In [21], the work proposed in [21] has been extended, and a multi-objective optimization consisting of delay as well as the number of gates have been considered. In [23, 24], 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 [23, 24] cannot be considered as a complete set. In [25], 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, the number of NOT gates) is proposed, which can be used for synthesis of three, four, or higher input functions. The concept of Majority Specification Matrix (MSM) is introduced and employed. Furthermore, the synthesis flow is considered for synthesizing multi-output functions. To compare the suggested method with other ones, benchmarks in [22] and MCNC benchmark circuits are used.

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

2. Background material

In this section, basic concepts in QCA technology such as Quantum-dot cellular automata and QCA devices are explained.

2.1. Quantum-dot cellular automata

A standard QCA cell (Figure 1) is composed of four dots located at the corners of a square. Two free electrons can tunnel to any quantum-dot within the cell [26]. 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 Figure 2. These configurations are denoted as cell polarization $P = +1$ (binary ‘1’ state) and $P = -1$ (binary ‘0’ state).

![Figure 1: a) Structure of a QCA cell with four quantum dots. b) QCA cell with two different polarizations.](image)

2.2. QCA devices

In this sub-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° to each other take on opposing polarization. A QCA majority gate can perform 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.

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

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:

\[ M(A, B, 0) = AB, \quad M(A, B, 1) = A + B. \]  

(2)

Figure 2 demonstrates a QCA wire, inverter gate, and majority gate, respectively.

![QCA Wire](image1.png)  
![Inverter Gate](image2.png)  
![Majority Gate](image3.png)

**Figure 2:** Representation of a) QCA wire b) inverter gate c) QCA majority gate.

### 3. Proposed method

The suggested synthesis method is based on the creation of Majority Specification Matrix (MSM). In the mentioned matrix, all the input states of a majority function are placed in each of columns. In fact, 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 figure 3, the dimensions of the matrix are \(8 \times 8\), and it can be considered as a regular matrix. More details on this topic can be found in [27-29]. In this matrix, binary number of each of the columns 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 is related to a NOT gate in the input of majority gate with a certain order.

The following feature can be specified in the mentioned matrix:

- Specification of output of pairs \((4, 5), (3, 6), (2, 7),\) and \((1, 8)\) are complementary.
- With respect to each of the two non-complementary columns, there are exactly two input states with 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 will be present:

\[
\begin{align*}
\text{Maj}(a, b, c) + \text{Maj}(a, b, c') &= \text{Maj}(a, b, c + c') = \text{Maj}(a, b, 1), \\
\text{Maj}(a, b, c) \times \text{Maj}(a, b, c') &= \text{Maj}(a, b, c \times c') = \text{Maj}(a, b, 0).
\end{align*}
\]  

(3)

- Changing the order of input variables does not change the specification function in each column.
In the following sections, the application of MSM for synthesis of Boolean functions will be elaborated on.

3.1 Three-input Boolean functions

This sub-section explains the application of MSM for the synthesis of three-input Boolean logic functions. In three-input functions, 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 numbers and the minimum number of NOT 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. At 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, the numbers of additional ones are removed, and minterms with additional zeros are converted to ones, respectively. Following this step, for each part K-map is used for further simplification of the final expression. In figure 4 an overall schematic of Method 1 is shown.

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

Table 1: The impact of applied AND/OR between pairs (applied majority gate, $F_{AND}$) then (applied majority gate, $F_{OR}$), respectively.

| Output function | Majority Gate applied | $F_{AND}$ | $F_{OR}$ |
|-----------------|-----------------------|-----------|---------|
| 0               | 1                     | 0         | 0       |
| 0               | 0                     | $X$       | 0       |
| 1               | 0                     | $X$       | 1       |
| 1               | 1                     | 1         | $X$     |

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

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

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.

Table 2: Representation of specification function related to Example 1 with 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                     | $X$       |

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. With 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 3: Simplification of $F_{AND}$ created with Karnaugh map.

| $bc$ | 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 have a value of one. Hence, one
possible majority expression for \( F_{\text{AND}} \) can be obtained as follows:
\[
F_{\text{AND}} = a + b + c = \text{Maj}(a, \text{Maj}(b', c, 1), 1).
\]  
(5)

The final result is provided in expression 8.
\[
F = \text{Maj}(\text{Maj}(a', b, c'), F_{\text{AND}}, 0) = \text{Maj}(\text{Maj}(a', b, c'), \text{Maj}(a, \text{Maj}(b', c, 1), 1)).
\]  
(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, in doing so, one of the majority gates would be removed. This method also can be used for proposed methods in the next sections.

3.1.2. Base of Method 2
In the second method, a majority gate with three-inputs \( f_1, f_2, f_3 \) are employed \((F = \text{Maj}(f_1, f_2, f_3))\); then, expressions related to the three functions are obtained. In figure 5 an overall schematic of Method 2 is shown.

In this method, same as Method 1, at first the column of MSM, which is more similar to the function specification, is selected \((F_1)\), and then the function of \( F_2 \) is obtained according to Table 4.

Table 4: Obtaining \( F_2 \) function according to \( F_1 \) and main function.

| Main Function | \( F_1 \) Majority Gate selected | \( F_2 \) |
|---------------|---------------------------------|---------|
| 0             | 0                               | \( X \) |
| 0             | 1                               | 0       |
| 1             | 0                               | 1       |
| 1             | 1                               | \( X \) |

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 \( F_2 \) would 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 or equal to two, then the value of output of majority function would be one; otherwise it would 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 values of minterms of function \( F_3 \) can be obtained considering Table 4. The following example illustrates the idea:

Example 2: Consider specification \( F(a, b, c) = (0, 2, 3, 6) \) defined as the first and the second columns of Table 5.
Table 5: Representation of specification function related to Example 2 with the application of Method 2.

| a | b | c | F | F(2) = Maj(a', b, c') | F2 | F3 |
|---|---|---|---|---------------------|---|---|
| 0 | 0 | 0 | 1 | 1                   | X = 0 | 1 |
| 0 | 0 | 1 | 0 | 0                   | X = 0 | X = 1 |
| 0 | 1 | 0 | 1 | 1                   | 0   | 0   |
| 0 | 1 | 1 | 1 | 1                   | X = 0 | 1 |
| 1 | 0 | 0 | 0 | 0                   | X = 1 | 0 |
| 1 | 0 | 1 | 0 | 0                   | X = 1 | 0 |
| 1 | 1 | 0 | 1 | 1                   | X = 1 | X = 0 |
| 1 | 1 | 1 | 0 | 0                   | X = 1 | 0 |

In first, the column of MSM with the least difference from main function is selected \((F(2) = Maj(a', b, c'))\). Then, the value of function \(F_2\) is obtained according to Table 4 and K-map presented in table 6 as follows \((F_2 = a)\).

Table 6: Representation of K-map applied for simplification of \(F_2\) function.

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

Value of function \(F_3\) is obtained considering \(F_2\) and Table 4; for example, the value obtained for \(F_2\) in state (000) is zero, which is not equivalent to value \(F_1\) and main function. Hence, value \(F_3\) would be equal to the value of main function. Furthermore, the values of \(F_2, F_1\), and the main function in state (001) 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.

Table 7: Applying of K-map for simplification of \(F_3\) function.

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

Logic expression for \(F_3\) is as follows:

\[
F_3 = a'(b' + c) = Maj(a', Maj(b', c, 1), 0) \tag{7}
\]

The total logic expression for main function \((F)\) is:

\[
F = Maj(Maj(a', b, c'), a, Maj(a', Maj(b', c, 1), 0)) \tag{8}
\]
3.1.2.1. Another structure of Method 2

Also, in some of the functions, combination of the first (AND, OR) and the second methods can lead to better results. For example, consider function \( F(a, b, c) = (3, 4, 6) \) as shown in Table 8.

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

| a | b | c | F | \( F_1 = \text{Maj}(7) = \text{Maj}(a, b, c) \) | \( F_{OR} \) | \( F_{tot} \) | \( F_2 \) | \( F_3 \) |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | \( X = 0 \) | \( X \) |
| 0 | 0 | 1 | 0 | 0 | 0 | \( X = 0 \) | \( X \) |
| 0 | 1 | 0 | 0 | 0 | 0 | \( X = 0 \) | \( X \) |
| 0 | 1 | 1 | 1 | 1 | 0 | \( X = 0 \) | 1 |
| 1 | 0 | 0 | 1 | 0 | 1 | \( X = 0 \) | 1 |
| 1 | 0 | 1 | 0 | 1 | 1 | \( X = 0 \) | 0 |
| 1 | 1 | 0 | 1 | 1 | 1 | \( X = 0 \) | 1 |
| 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 0 |

At first, the most similar column of MSM to the main function is selected (\( F_1 = \text{Maj}(a, b, c) \)). With the selection of function \( F_1 \) as presented in Table 8 and the examination of K-map, it can be observed that to apply the OR operation between \( F_1 \) and \( F_{OR} \) (columns 3 and 4 in Table 8), all the needed 1’s states should be created without adding extra majority function. It means that the logic expression for \( F_{OR} \) is \( F_{OR} = a \), as shown in Figure 6.

![Figure 6: Applying of OR operation to F1 function.](image)

The obtained result has been shown in column 5 (\( F_{tot} = \text{Maj}(\text{Maj}(a, b, c), a, 1) \)). Hence, with consideration of Table 4, comparison of values related to \( F_{tot} \) and the main function, and 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, with the application of K-map, the logic function is simplified (as shown in Table 9). Thus, the logic expression for \( F_3 \) is:

\[
F_3 = a' + c' = \text{Maj}(a', c', 1)
\]  

(9)

Table 9: Applying of K-map for simplification of \( F_3 \) function.

| bc | 0 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 | \( X = 1 \) | \( X = 1 \) | 1 | \( X = 1 \) |
| 1 | 1 | 0 | 0 | 1 |
In result, the total logic expression obtained for main function \((F)\) is:

\[
F = Maj(Maj(Maj(a, b, c), a, 1), 0, Maj(a', c', 1))
\]  

(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 modify the obtained results. In this method, after applying Method 2 to the specification function and obtaining functions of \(F_2\) and \(F_3\), K-maps related to \(F_2\) and \(F_3\) are simultaneously reviewed. 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 or equal to two, 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 are marked that value of function is one in them. Due to the mentioned feature for 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 \(F_2\) is 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 changed in response to the rules of group 1 and 2. This method as a search method is continued until the best square in the k-maps is obtained.
- In each majority gate, there is a bellow expression. With the application of that expression, the number of NOT gates in the general expression can be decreased.

\[
Maj(a', b', c') = Maj(a, b, c)'
\]  

(11)

Generally, only states of \(F_2\), which are “don’t care”, are considered. Then, the states generated in \(F_3\) which do 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 \(f, g, u, d\) are Boolean functions and the following expression exists between them: 
  \[ y = u(f g' + g d) \]  
  where \(g'\) is the complement of \(g\). Then the equivalent majority expression is as the following expression: 
  \[ y = M(u, M(f, g', 0), M(g, d, 0)) \]

Prove: By extending equivalent majority expression, we have:

\[
y = u(f g' + g d) + (f g') \times (g d) = u(f g' + g d).
\]

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

In this sub-section, for the synthesis of four- and higher-input Boolean functions, it must be considered that the majority gate has 3-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.

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

1. Taking into account that the majority gate has 3-inputs; thus, for four-input functions with consideration of the inputs 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 inputs of 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.

\[
\text{Maj}(b, c, d), \text{Maj}(a, b, c), \text{Maj}(a, c, d), \text{Maj}(a, b, d)
\]

(12)

For each of the above-mentioned permutations, an MSM must be created according to Figure 3. For creation of MSM with b, c, and d inputs, two matrices of MSM can be created in the 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 creation of MSM in the other states, each row of MSM created in Figure 3 must move to two new rows for four-input functions. For example, consider the \( \text{Maj}(a, b, c) \) state. For creation of the mentioned state, the majority function \( (a, b, c) \) must shift variables \( (a, b, c) \) to the left in comparison to state \( (b, c, d) \). In Table 10, this method has been shown. In fact, variable d in the main permutation \( (b, c, d) \) is as a “don’t care” variable for new permutation \( (a, b, c) \).

Table 10: Creation of new states in four-input functions according to the three-input functions.

| Old rows | New permutation of rows | Old rows | New permutation of rows |
|----------|-------------------------|----------|-------------------------|
| b c d    | a b c d                 | b c d    | a b c d                 |
| 0 0 0    | 0 0 0 0                 | 1 0 0    | 1 0 0                   |
|          | 0 0 0 1                 |          | 1 0 1                   |
| 0 0 1    | 0 0 1 0                 |          | 1 0 1                   |
|          | 0 0 1 1                 |          | 1 0 1                   |
| 0 1 0    | 0 1 0 0                 | 1 1 0    | 1 1 0                   |
|          | 0 1 0 1                 |          | 1 1 0                   |
| 0 1 1    | 0 1 1 0                 | 1 1 1    | 1 1 1                   |
|          | 0 1 1 1                 |          | 1 1 1                   |

2. 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 of \( F_2 \) or \( F_3 \) functions leads to creation of complex specification functions, as a result many majority gates will be created in each function. Hence, for synthesis of \( F_2 \) or \( F_3 \) functions, each of them can be considered as a main function. Then, in the process of applying Method 2 explained in three-input functions section, the mentioned functions can be synthesized once more. This method can be repeatedly carried out to determine its acceptable level. In figure 7 an overall schematic of this method is shown.

![Figure 7: An overall schematic of Method 3.](image)

For creation of MSM for functions larger than three-inputs, the methods explained in the previous section as well as the above-mentioned methods can be used. In the following examples, the above-mentioned methods have been
Example 4. Consider function \( F = (9,11,14) \) defined as the first and the second columns of Table 11.

| \( a \) | \( b \) | \( c \) | \( d \) | \( F \) | \( F_1 = M(7) = M(a, b, d) \) | \( F_2 \) | \( F_3 \) |
|---|---|---|---|---|---|---|---|
| 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 | 1 | 0 |

According to Method 2 explained in three-input functions, at first the column of MSM that is more similar 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 logic function \( F_2 \) is zero (\( F_2 = 0 \)), and the logic function \( F_3 \) is obtained with the application of K-map as shown in Table 12:

| \( cd \) | \( ab \) | 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 | \( X = 1 \) |

Boolean logic function \( F_3 \) is:

\[ F_2 = b' + c'd' = \text{Maj}(b', \text{Maj}(c, d'), 0), 1 \]  \( (13) \)

With the implementation of the post-processing method presented in three-input functions section, specification functions of \( F_2 \) and \( F_3 \) can be exchanged as shown in Figure 8. States of main specification function, which has the value of one, is presented by gray colour in Figure 7. Moreover, values of states 0000 and 1000 have been changed to zero in Figure 8(b). As the mentioned states in the main specification 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, 10), which has been exchanged to the same column in Figure 8(a) (\( F_2 \)).
Then, states (1001, 1011) are fix thus, among of other states, states (0000, 0100) must be zero until it creates a square in K-map. Logic expressions for F2 and F3 are:

\[ F_2 = c'd' = Maj(c, d', 0), \quad F_3 = b'd = Maj(b', d, 0). \]  

The total logic function is:

\[ F = Maj(Maj(a, b, d), Maj(c, d', 0), Maj(b', d, 0)) \]  

Example 6. Consider specification function \( F = (3, 4, 7, 15) \) is defined as the first and the second columns of Table 13; in this example, using of tree method is explained.

Table 13: Specification function used in Exp. 6 and calculation \( F_2 \) 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 | \( X = 1 \) | \( 1 \) |
| 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 | \( X = 0 \) | \( X \) |
In first, most similar column to main function is selected; that here, \( \text{Maj}(0, c, d) \) is combined from \( \text{Maj}(b, c, d) \) and \( \text{Maj}(b', c, d) \). Then, the logic function \( F_2 \) according to K-map shown in Table 14 is obtained. \( (F_2 = b) \)

Table 14: K-map use for calculation \( F_2 \)

| cd | ab | 00  | 01  | 11  | 10  |
|----|----|-----|-----|-----|-----|
| 00 | X  | X   | X   | X   |
| 01 | 1  | X   | X   | X   |
| 11 | X  | X   | X   | X   |
| 10 | X  | X   | X   | X   |

For obtaining of specification function \( F_3 \) as tree method, its value is considered as the main function and is shown in Table 15.

Table 15: Consider \( F_2 \) function as main function for using to tree method.

| a   | b   | c   | d   | \( F_2 \) | \( F_{3,1} = \text{M}(a', b', c') \) | \( F_{3,2} \) | \( F_{3,3} \) |
|-----|-----|-----|-----|--------|----------------|--------|--------|
| 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   | X      | 1              | X = 0  | X = 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  |

As it is shown in Table 15, function \( F_3 \) as main function is considered. It is important to note that the states of “do not care” in Function \( F_3 \) in calculation of specification functions of \( F_{3,1}, F_{3,2} \) and \( F_{3,3} \) are do not care and it does not matter that their values be zero or one. The first, the most similar column to main function is selected (\( \text{M}(a', b', c') \)); logic functions of \( F_{3,2} \) and \( F_{3,3} \) are:

\[
F_{3,2} = 0, \quad F_{3,3} = d' + c = \text{Maj}(d', c, 1).
\] (16)
Then, specification functions of $F_{3,2}$ and $F_{3,3}$ are obtained by using the post-processing method as shown in Figure 9.

![Table](#)

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

Figure 9: (a) K-map is used for function $F_{3,2}$, (b) K-map is used for function $F_{3,3}$. Apply post-processing method to $F_{3,2}$ and $F_{3,3}$ for more simplifying.

New Boolean logic functions $F_{3,2}$ and $F_{3,3}$ are:

$$F_{3,2} = b', \quad F_{3,3} = d'. \quad (17)$$

The total specification function is:

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

4. A synthesis flow for multi-output functions

Having the proposed approaches from the previous section available, they can be combined to an extended synthesis flow that could be used for multi-output functions. Figure 10 illustrates this flow. As it is shown in this figure, in first, according to the numbers of inputs of main function, MSMs are created, then for each of outputs of function ($f_i$) and complementary it, the most similar column of MSM or inputs of function is selected; as well, it could be selected from combination of columns in MSM that led to AND/OR functions, then, for each of selected columns, the proposed methods in previous sections are applied. Afterwards, the obtained results are saved. In addition, conventional K-map method, is applied and its result is saved. Because of that in some functions, expression obtained from this method is simpler. Then, among of results obtained for each of outputs, for arriving to the most common expressions, in row 10 of the algorithm, from expressions obtained in outputs for synthesis the other outputs are used, then results based on objective priority gate counts and gate levels are ordered, then results to the most common terms are selected. Finally, for reducing of the number of NOT gates in final expressions, row 12 of algorithm is applied.
Input: a given specification of multi-output function
Output: a synthesized circuit to majority gate

1. Creation of MSM according to the number of inputs of function
2. For each of \( f_i \) (\( i = 1:m \) m: the number of outputs of function)
3. \{ For each of MSMS created
4. \{ Find most similar column to specification of main function in MSM
5. Find most similar column according to features of combination of columns in MSM.
6. Find most similar specification between main function and columns of inputs.
7. Do rows 4 and 5 for complementary function (\( f_i \)), again.
8. Apply the all methods explained in previous sections to rows of (4) and (5) from algorithm and save all results.
9. Apply method of conventional K-map to main function and save the result.
10. Use from majority expression created in each of \( f_i \) for synthesis other \( f_j \) (\( j \neq i \))
11. Select results to the most common expressions between outputs.
12. For reducing the number of NOT gate, use from the following feature
   \[ \text{Maj}(a', b', c)' = \text{Maj}(a, b, c) \]

Figure 10: Synthesis flow for synthesis of multi-output functions.

5. Results and Comparison
In this section, an overall comparison between the multi-objective genetic programming [22], the 20 MCNC benchmark circuits and the proposed method are illustrated. The results have been shown in Figure 11 and Table 16. As it is shown in Figure 11, columns of NOI, NOM and NTG are considered as number of inverter gates, number of majority gates and number of total Gates, respectively. The Boolean functions obtained have been shown in final column. For each of function outputs based on the two following features have been selected:

1. In each of outputs, there are lowest number of gates, gate levels and the number of NOT gates, respectively.
2. In each of outputs, there are most number of common parts.

For each of outputs, the common parts of the corresponding synthesized circuit have been underlined. Also, constant inputs in the results shown in final column for comparison to [22] have been changed to value 1. In some functions, must been trade off between the number of gates and the number of common parts; for example, in specification function \( F = (2,6,10,11,14) \) the following expressions have obtained (this function selected from the multi-output function shown in figure 11):

\[
F_1 = \text{Maj}(\text{Maj}(a, b', 0), d', 1), c, 0) \\
F_2 = \text{Maj}(\text{Maj}(1', c, d), b, \text{Maj}(\text{Maj}(a, b, c), b, d)', 1, c)
\]

(19)

That between them, \( F_2 \) has been selected because in it, the numbers of common parts are caused reduction of the numbers of total gates. (According to other outputs presented in Figure 11) Our result illustrate that our proposed method is as well as our Meta-heuristics alike genetic programming even it is better in some of results. In Table 16 an overall comparison between the best existing majority logic synthesis method [30] and the proposed method is demonstrated. Also, in this table the proposed method is compared with the method presented in [31]. the first column lists the names of benchmarks. The columns under the title “Method [31]” and “Method [30]” show the results for the corresponding benchmarks obtained from [31] and [30] in terms of the number of levels and gates. Also, the number of NOT gates for [30], as proposed method in [31] have not been reported the number of NOT gates. The “Reduction %” columns compare the proposed method with the methods in [31] and [30] and give the percentage reductions. This table illustrates that there is an average reduction of 14.9% in the number of levels. At the same time, the number of gates is reduced by 31.6% when compared to the method in [31]. When compared to [30], the average reduction in levels is 10.5% and the reduction in gate counts is 16.8% and also the reduction in NOT gates is 33.5%. Results show that our proposed method outperforms the existing methods.
6. Conclusion
In this paper, a comprehensive and multi-objective method for generating optimal majority expression has been presented. In this method by using a Majority Specification Matrix (MSM) and synthesis flow presented for multi-output specification functions, can arrive to optimal majority expression. The proposed method was applied to 20 MCNC benchmarks and meta-heuristic methods as genetic programming. Obtained results demonstrate the proposed method outperforms the existing ones.
| function | specification | [22] | Our Proposed Method | The Best Circuit |
|----------|---------------|------|---------------------|------------------|
| f1       | (2,4,6)       | 3    | 4                   | 7                | M(M(a,b,c'),0,c') |
| f2       | (0,1,3,6)     | 3    | 4                   | 7                | M(M(a,M(a,b,c'),1'),1,M(a,b,1')) |
| f3       | (0,3,6)       | 4    | 4                   | 8                | M(a',b,c'),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 |
| f1       | (1,4,5,7)     | 1    | 1                   | 2                | M(a,b',c)        |
| f2       | (3,4,6)       | 3    | 4                   | 7                | M(a,b,c,M(a,c',1),M(a,c',1')) |
| f3       | (0,2,5,6)     | 3    | 4                   | 7                | M(a,b',c'),M(a,c,1),M(c,b',1')) |
| f4       | (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 |
| f1       | (0,3,6,7,15)  | 3    | 5                   | 8                | 4                | 5                | 9                | M(a,b,M(c,d,1),M(c,M(b,d,1)',1,1')) |
| f2       | (9,11,14)     | 4    | 4                   | 8                | 4                | 4                | 6                | M(a,b,d,1,M(a,b,M(c,d,1'))) |
| f3       | (8,10,11,14,15) | 1 | 3 | 4 | 2 | 3 | 5 | 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 |
| f1       | (3,4,7,15)    | 3    | 4                   | 7                | 2                | 4                | 6                | M(M(1',c,d),b,M(a,b,c,b,d')) |
| f2       | (1,3,4,9,13,15) | 4 | 5 | 9 | 3 | 5 | 8 | M(M(a,b',1,d,1'),b,M(a,b,c,b,d')) |
| f3       | (3,6,7,11,13,14,15) | 2 | 3 | 5 | 2 | 3 | 5 | M(M(a,b,1,d,1'),b,c) |
| f4       | (2,6,10,11,14) | 5    | 5                   | 10               | 3                | 5                | 8                | M(M(1',c,d),b,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 |

Figure 11: Comparison proposed method to method multi-objective genetic programming presented in [22].
Table 16: Comparisons with methods [31] and [30].

| Benchmarks | Method [31] | Method [30] | Proposed method | Reduction% VS. [31] | Reduction% VS. [30] |
|------------|-------------|-------------|-----------------|---------------------|---------------------|
|            | Level | Gate | Level | Gate | Inverters | Level | Gate | Inverter | Level | Gate | Inverter | Level | Gate | Inverter |
| 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% |
| i1         | 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% |
| pcle       | 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

|                        | Reduction% VS. [31] | Reduction% VS. [30] |
|------------------------|---------------------|---------------------|
|                        | 14.9%               | 31.6%               |
|                        | 10.5%               | 16.8%               |
|                        | 33.5%               |
Appendix

In this section, the obtained circuits of 20 MCNC benchmarks are illustrated (Table A1).

Table A1: illustrating of the obtained circuits of 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),M(a,b',1),1), d=c', g=c' |
| cm82a      | f=M(M(a,b,c)',M(a,b',c),b), g=M(M(a,b,c),d,e)',M(a,b,c),d',e), d=M(M(a,b,c),d,e) |
| majority   | f=M(d,M(a,b,c),M(e,c,0)),M(e,c,1),0),1) |
| 9symml     | x2=|<52=>M(M(M(7815),M(7819),M(f(M(a,b,c),M(a,b',c),b),g=M(M(a,b,c),d,e)',M(a,b,c),d',e),d), h=M(M(a,b,c),d,e) |
| 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',0),M(d,j,0)),M(k,M(f,j',0),M(h,j,0))),1) |
| cm85a      | L=M(M(M(h,i,M(j,k,0),f,g),d',e),b,0),a,1), n=M(M(b,0,M(M(h,i',M(j,k',0)),f,g'),d',e'),c,1), m=M(b,0,M(M(M(h,i,M(j,k,0),f,g),d',e),M(M(h,i',M(j,k',0)),f,g'),d',e'),1)) |
| cm151a     | <n>=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',M(k,M(e,i',0),M(f,i,0)),M(k',M(a,i',0),M(b,i,0))), n=M(n')
\[ <V27_1> = M(M((in_{V27_0}), (in_{V29_0}), 0), M([33], M(V8_0, V9_0), M(V8_0', (V9_0'), 0)), 1), \]
\[ <V27_2> = M(M((in_{V27_0}), (in_{V29_0}), 0), M(M(V7_1, V7_2, 1), M(V7_3, V7_4), 1), M(M(V7_5, V7_6, 1), V7_7, 1)), 0), M(V8_0, V9_0, [33], 0), 1), \]
\[ <V27_4> = M(V8_0, V9_0), (V22_2), 1), \]
\[ <V28_0> = M(V10_0, M(V8_0, [33], 0), 1), \]
\[ <V30_0> = M(V18_0, V22_5, 1), \]
\[ <V31_0> = V11_0, \]
\[ [33] = M(M(V7_1, V7_2, 1), M(V7_3, V7_4), 1), M(M(V7_5, V7_6, 1), V7_7, 1)), 0), \]
\[ decod = M(M(a, d, 0), M(e, M(b, c), 0), 0), \]
\[ g = M(M(a, d', 0), M(e, M(b, c), 0), 0), \]
\[ h = M(M(a, d, 0), M(e, M(b, c'), 0), 0), i = M(M(a, d', 0), M(e, M(b, c'), 0), 0), \]
\[ j = M(M(a, d, 0), M(e, M(b, c'), 0), 0), k = M(M(a, d', 0), M(e, M(b, c'), 0), 0), \]
\[ L = M(M(a, d, 0), M(e, M(b, c'), 0), 0), M = M(M(a, d), M(e, M(b, c'), 0), 0), \]
\[ N = M(M(a, d), M(e, M(b, c'), 0), 0), O = M(M(a, d'), M(e, M(b, c'), 0), 0), \]
\[ P = M(M(a, d), M(e, M(b, c'), 0), 0), Q = M(M(a, d'), M(e, M(b, c'), 0), 0), \]
\[ R = M(M(a, d), M(e, M(b, c'), 0), 0), S = M(M(a, d'), M(e, M(b, c'), 0), 0), \]
\[ T = M(M(a, d), M(e, M(b, c'), 0), 0), U = M(M(a, d'), M(e, M(b, c'), 0), 0), \]
\[ tcon = M(a, M(i, k, 1), M(i', k, 0)), b0 = M(b, M(i, L, 1), M(i', L, 0)), \]
\[ c0 = M(c, M(i, m, 1), M(i', m, 0)), d0 = M(d, M(i, n, 1), M(i', n, 0)), \]
\[ e0 = M(e, M(i, o, 1), M(i', o, 0)), f0 = M(f, M(i, p, 1), M(i', p, 0)), \]
\[ g0 = M(g, M(i, q, 1), M(i', q, 0)), h0 = M(h, M(i, r, 1), M(i', r, 0)), \]
\[ v = n, w = o, x = p, y = q, z = r \]
References

[1] International technology roadmap for semiconductors (ITRS). [Online]. Available: http://www.itrs.net
[2] K. Hennessy and C. S. Lent, "Clocking of molecular quantum-dot cellular automata," Journal of Vacuum Science & Technology B, vol. 19, pp. 1752-1755, 2001.
[3] C. S. Lent, B. Isaksen, and M. Lieberman, "Molecular quantum-dot cellular automata," Journal of the American Chemical Society, vol. 125, pp. 1056-1063, 2003.
[4] C. S. Lent and P. D. Tougaw, "A device architecture for computing with quantum dots," Proceedings of the IEEE, vol. 85, pp. 541-557, 1997.
[5] P. D. Tougaw and C. S. Lent, "Logical devices implemented using quantum cellular automata," Journal of Applied Physics, vol. 75, pp. 1818-1825, 1994.
[6] T. Oya, T. Asai, T. Fukui, and Y. Amemiya, "A majority-logic nanodevice using a balanced pair of single-electron boxes," Journal of nanoscience and nanotechnology, vol. 2, pp. 333-342, 2002.
[7] T. Oya, T. Asai, T. Fukui, and Y. Amemiya, "A majority-logic device using an irreversible single-electron box," Nanotechnology, IEEE Transactions on, vol. 2, pp. 15-22, 2003.
[8] H. Fahmy and R. A. Kiehl, "Complete logic family using tunnelingphase-logic devices," in Proc. Int. Conf. Microelectron, 1999, pp. 22-24.
[9] C. S. Lent, P. D. Tougaw, and W. Porod, "Bistable saturation in coupled quantum dots for quantum cellular automata," Applied Physics Letters, vol. 62, pp. 714-716, 1993.
[10] P. D. Tougaw, C. S. Lent, and W. Porod, "Bistable saturation in coupled quantum-dot cells," Journal of Applied Physics, vol. 74, pp. 3558-3566, 1993.
[11] G. Bourianoff, M. Brillouet, R. K. Cavin III, T. Hiramoto, J. A. Hutchby, A. M. Ionescu, et al., "Nanoelectronics Research for Beyond CMOS Information Processing," Proceedings of the IEEE, vol. 98, pp. 1986-1992, 2010.
[12] J. Hutchby, G. Bourianoff, V. V. Zhirnov, and J. E. Brewer, "Extending the road beyond CMOS," Circuits and Devices Magazine, IEEE, vol. 18, pp. 28-41, 2002.
[13] H. Miller and R. Winder, "Majority-logic synthesis by geometric methods," IEEE Transactions on Electronic Computers, vol. 1, pp. 89-90, 1962.
[14] R. Zhang, K. Walus, W. Wang, and G. Jullien, "A method of majority logic reduction for quantum cellular automata," Nanotechnology, IEEE Transactions on, vol. 3, pp. 443-450, 2004.
[15] M. Karnaugh, "The map method for synthesis of combinational logic circuits,' Commun," ed: Electronics, 1953.
[16] S. B. Akers Jr, "Synthesis of combinational logic using three-input majority gates," in Switching Circuit Theory and Logical Design, 1962. SWCT 1962. Proceedings of the Third Annual Symposium on, 1962, pp. 149-158.
[17] K. Walus, G. Schulhof, G. Jullien, R. Zhang, and W. Wang, "Circuit design based on majority gates for applications with quantum-dot cellular automata," in Signals, Systems and Computers, 2004. Conference Record of the Thirty-Eighth Asilomar Conference on, 2004, pp. 1354-1357.
[18] Z. Huo, Q. Zhang, S. Haruchaneoengra, and W. Wang, "Logic optimization for majority gate-based nanoelectronic circuits," in Circuits and Systems, 2006. ISCAS 2006. Proceedings. 2006 IEEE International Symposium on, 2006, pp. 4 pp.-1310.
[19] M. Bonyadi, S. Azghadi, N. Rad, K. Navi, and E. Afjei, "Logic optimization for majority gate-based nanoelectronic circuits based on genetic algorithm," in Electrical Engineering, 2007. IEECE'07. International Conference on, 2007, pp. 1-5.
[20] M. Houshmand, S. H. Khayat, and R. Rezaei, "Genetic algorithm based logic optimization for multi-output majority gate-based nano-electronic circuits," in Intelligent Computing and Intelligent Systems, 2009. ICIS 2009. IEEE International Conference on, 2009, pp. 584-588.
[21] M. Houshmand, R. R. Saleh, and M. Houshmand, "Logic minimization of QCA circuits using genetic algorithms," in Soft Computing in Industrial Applications, ed: Springer, 2011, pp. 393-403.
[22] R. Rezaee, M. Houshmand, and M. Houshmand, "Multi-objective optimization of QCA circuits with multiple outputs using genetic programming," Genetic Programming and Evolvable Machines, vol. 14, pp. 95-118, 2012.
[23] P. Wang, M. Niamat, and S. Vemuru, "Minimal majority gate mapping of 4-variable functions for quantum cellular automata," in Nanotechnology (IEEE-NANO), 2011 11th IEEE Conference on, 2011, pp. 1307-1312.
[24] P. Wang, M. Niamat, and S. Vemuru, "Minimal Majority Gate Mapping of Four-Variable Functions for
Quantum-Dot Cellular Automata," *Nanoelectronic Device Applications Handbook*, pp. 263-280.

[25] P. Wang, M. Niamat, and S. Vemuru, "4-variable Standard Function Majority Gate Logic Synthesis Using Graph Isomorphism," in *Field-Coupled Nanocomput. Workshop, Tampa, FL, USA*, 2013.

[26] I. Amlani, A. O. Orlov, R. K. Kummamuru, G. H. Bernstein, C. S. Lent, and G. L. Snider, "Experimental demonstration of a leadless quantum-dot cellular automata cell," *Applied Physics Letters*, vol. 77, pp. 738-740, 2000.

[27] R. Shedsale and N. Sarwade, "A Review Of Construction Methods For Regular Ldpc Codes," *Indian Journal of Computer Science and Engineering (IJCSE) Vol*, vol. 3, pp. 380-385, 2012.

[28] B. Ammar, B. Honary, Y. Kou, J. Xu, and S. Lin, "Construction of low-density parity-check codes based on balanced incomplete block designs," *IEEE Transactions on Information Theory*, vol. 50, pp. 1257-1268, 2004.

[29] B. Honary, S. Lin, E. M. Gabidulin, J. Xu, Y. Kou, A. Moinian, *et al.*, "On construction of low density parity check codes," presented at the *The 2nd International Workshop: Signal Processing for Wireless Communications 2004 (SPWC 2004)*, London, 2004.

[30] K. Kong, Y. Shang, and R. Lu, "An optimized majority logic synthesis methodology for quantum-dot cellular automata," *Nanotechnology, IEEE Transactions on*, vol. 9, pp. 170-183, 2010.

[31] R. Zhang, P. Gupta, and N. K. Jha, "Majority and minority network synthesis with application to QCA-, SET-, and TPL-based nanotechnologies," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 26, pp. 1233-1245, 2007.