Scientific Research Publishing

**Engineering, 2017, 9, 890-915**
http://www.scirp.org/journal/eng
ISSN Online: 1947-394X
ISSN Print: 1947-3931

# A Comparative Study of Majority/Minority Logic Circuit Synthesis Methods for Post-CMOS Nanotechnologies

**Amjad Almatrood, Harpreet Singh**

Department of Electrical and Computer Engineering, Wayne State University, Detroit, USA
Email: amjad.almatrood@wayne.edu

**How to cite this paper:** Almatrood, A. and Singh, H. (2017) A Comparative Study of Majority/Minority Logic Circuit Synthesis Methods for Post-CMOS Nanotechnologies. *Engineering*, **9**, 890-915.
https://doi.org/10.4236/eng.2017.910054

**Received:** August 10, 2017
**Accepted:** October 24, 2017
**Published:** October 27, 2017



## Abstract

The physical limitations of complementary metal-oxide semiconductor (CMOS) technology have led many researchers to consider other alternative technologies. Quantum-dot cellular automate (QCA), single electron tunneling (SET), tunneling phase logic (TPL), spintronic devices, etc., are some of the nanotechnologies that are being considered as possible replacements for CMOS. In these nanotechnologies, the basic logic units used to implement circuits are majority and/or minority gates. Several majority/minority logic circuit synthesis methods have been proposed. In this paper, we give a comparative study of the existing majority/minority logic circuit synthesis methods that are capable of synthesizing multi-input multi-output Boolean functions. Each of these methods is discussed in detail. The optimization priorities given to different factors such as gates, levels, inverters, etc., vary with technologies. Based on these optimization factors, the results obtained from different synthesis methods are compared. The paper also analyzes the optimization capabilities of different methods and discusses directions for future research in the synthesis of majority/minority logic networks.

## Keywords

Logic Design, Logic Optimization, Majority Logic Circuits, Post-CMOS Technologies

## 1. Introduction

The complementary metal-oxide semiconductor (CMOS) technology has played a vital role in constructing integrated systems for the past four decades. This technology has provided the requirements of implementing high-density, high-

speed and low-power very large scale integrated systems. However, the fundamental physical limits of this technology have been reached [1]. Many researches have introduced different nanotechnologies such as quantum-dot cellular automate (QCA) [2]-[7], single electron tunneling (SET) [8] [9], tunneling phase logic (TPL) [10], spintronic devices [11], and many other nanotechnologies. These nanotechnologies are being considered as possible replacements for CMOS technology and expected to provide further scaling down of feature sizes and other features of integrated systems. In CMOS technology, logic NAND, NOR and NOT gates are the basic units used to implement circuits. However, the post-CMOS nanotechnologies use logic majority and/or minority gates.

Traditional Boolean logic functions are simplified and expressed in two standard forms which are sum of products (SOP) and product of sums (POS) in terms of logic AND, OR and NOT gates. These standard forms are always produced using logic reduction methods that target CMOS technology. However, these methods are not efficient enough to produce simplified expressions in terms of logic majority or minority for post-CMOS nanotechnologies due to the complexity of multi-level majority and minority circuits. Since the function of minority gate is just the complement of majority gate, a minority logic network can be easily produced from its equivalent majority network. This process can be done by using De Morgan's theorem which is based on the use of inverters. Thus, by having an efficient majority logic synthesis method, both majority and minority logic networks can be obtained.

The history of research in majority logic synthesis dates back to the 1960s. Karnaugh-map (K-map) [12], reduced-unitized-table [13], and Shannon's decomposition principle [14] are some of these methods that were developed to synthesize majority logic network. However, these methods are suitable only for small networks because they are used to synthesize majority networks by hand. Other majority synthesis methods were introduced based on geometric interpretation of the three-variable Boolean functions to convert sum of products expressions into optimal majority logic networks [15] [16]. However, these methods can synthesize only up to three-variable Boolean functions. For synthesizing majority logic networks with more than three-variable, several approaches have been proposed based on different concepts [17]-[22]. Methods in [17] [18] are developed based on genetic algorithm [23] [24] and the concept of Boolean disjointness, respectively. Other approaches described in [19] [20] [21] [22] use a standard logic synthesis tool which is sequential interactive synthesis (SIS) [25] to decompose Boolean functions into three-feasible or four-feasible networks. The decomposed networks are then converted into their equivalent majority expressions based on different techniques. Recently, Wang *et al.* [26] have proposed a new comprehensive majority/minority synthesis method. This method also uses SIS tool. However, the decomposition methods used in this approach is developed to decompose input Boolean functions into both three-feasible and four-feasible networks. Based on a developed table that contains optimal equiva-

lent majority expressions for all four-variable Boolean functions, the decomposed networks are then converted into their corresponding majority expressions.

Even though many majority/minority logic network synthesis methods have been proposed, none of these methods can synthesize optimal majority/minority logic networks for all cases. As the purpose of this paper is to provide a review of the best synthesis methods, we only concentrate on the multi-input multi-output majority/minority logic networks synthesis methods and do not discuss the limited methods.

The rest of the paper is organized as follows. In Section 2, we discuss some of majority/minority-based post-CMOS nanotechnologies and the implementation of their logic devices. Section 3 describes the best available comprehensive majority/minority logic synthesis methods. Section 4 compares the synthesis methods described in Section 3 and discusses experimental results using these methods. The paper is concluded in Section 5.

## 2. Majority/Minority-Based Post-CMOS Nanotechnologies

In this section, we review some post-CMOS nanotechnologies and the implementation of their majority and/or minority logic devices.

### 2.1. Quantum-Dot Cellular Automata Technology

Quantum-dot cellular automata (QCA) technology is one of nanotechnologies that provide a new technique of computation information transformation. This technology uses a QCA majority gate as the basic device along with QCA wire and QCA inverter to implement logic circuits.

#### 2.1.1. QCA Cell

A QCA cell contains four quantum dots that are located at the corners of a square. By charging a cell with two free electrons, which tunnel between dots, there are only two states of electrons pairs that are energetically stable due to Coulombic interactions. The two configurations of electrical charges in a cell encode binary information. Each of these configurations has a different cell polarization. These polarizations are $P = +1$ and $P = -1$ which represent logic 1 and 0, respectively. **Figure 1** shows a QCA cell and its two possible electron configurations.

#### 2.1.2. QCA Devices

In QCA, a logic circuit is implemented using three primitive devices that are QCA wire, QCA inverter, and QCA majority logic gate. The construction of these devices is based on QCA cell which is the fundamental unit in QCA.

A QCA wire can be constructed by placing a group of cells next to each other as shown in **Figure 2(a)**. The binary signal propagates from the leftmost cell which is the input along the wire to the rightmost cell which is the output [27]. The direction of signal flow can be determined by the QCA clocks [28].

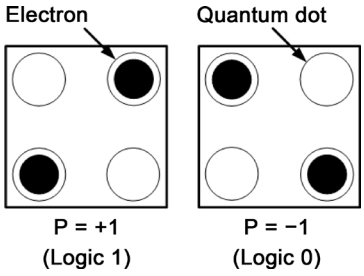


Figure 1. The possible elctron configurations of a QCA cell.

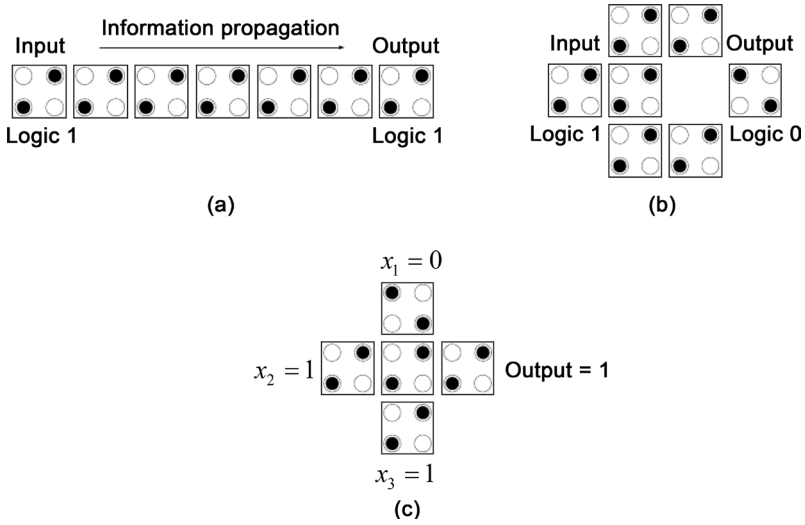


Figure 2. QCA devices: (a) QCA wire; (b) QCA inverter; (c) QCA majority gate.

By placing cells in a diagonal position, the polarizations of these cells will be reversed. Based on this characteristic, the QCA inverter can be constructed as shown in Figure 2(b).

The function of a QCA majority gate is a three-input logic function given in (1). The majority function is to produce an output logic 1 if two or more of the three inputs $(x_1, x_2, x_3)$ are 1. Otherwise, it produces an output logic 0. The layout of QCA majority gate is shown in Figure 2(c). As seen in the figure, a QCA majority gate is constructed of one cell surrounded by four cells, one in each side. Three of these cells are the gate inputs which are the upper, leftmost and lower cells. Based on the polarizations of the three input cells, the middle cell polarization is determined because it represents the lowest energy state. Then, the signal propagates to the rightmost cell which is the output cell.

$$M(x_1, x_2, x_3) = x_1x_2 + x_1x_3 + x_2x_3 \tag{1}$$

By forcing one of the inputs $(x_1, x_2, x_3)$ in a three-input majority gate to logic 0 or 1, the gate will perform as a two-input logic AND or a two-input logic OR function as given in (2) and (3), respectively.

$$M(x_1, x_2, 0) = x_1x_2 \tag{2}$$

$$M(x_1, x_2, 1) = x_1 + x_2 \tag{3}$$

## 2.2. Single Electron Tunneling Technology

In single electron tunneling (SET) technology, both majority and minority gates are used to implement logic circuits. A SET minority gate implements a three-input logic function given in (4). Since the minority function is just the complementary of majority function, it produces an output 0 if one or more of its inputs are 1. Otherwise, it produces an output 1.

$$m\left(x_1, x_2, x_3\right) = x_1'x_2' + x_1'x_3' + x_2'x_3' \tag{4}$$

**Figure 3(a)** shows a basic SET minority gate. It consists of three input capacitors, single-electron boxes (SEBs), and an output capacitor. The inputs of minority gate ($V_1$, $V_2$, and $V_3$) move through the input capacitors to form a voltage summing network. These capacitors produce the mean voltage of their inputs at node A. Based on the value of the mean voltage, an electron will tunnel through SEBs and make the voltage at node A negative. Otherwise, the voltage will remain positive. The negative and positive values represent logic 0 and 1, respectively.

By setting one of the three inputs of the minority gate as a logic 0 or 1, the gate implements a two-input logic NAND or two-input logic NOR gate, respectively [8]. The obtained functions are given by

$$m\left(x_1, x_2, 0\right) = x_1' + x_2' = \left(x_1x_2\right)' \tag{5}$$

$$m\left(x_1, x_2, 1\right) = x_1'x_2' = \left(x_1 + x_2\right)' \tag{6}$$

A SET majority gate is constructed of three input capacitors, a balanced pair of SEBs, three output capacitors as shown in **Figure 3(b)**. When the bias voltage (Vdd) increases, the electron tunneling occurs and results in either (0, 1) or (1, 0) stable voltage state. The (0, 1) state occurs and produces a positive value at node B if the majority of the inputs are 1. Otherwise, the (1, 0) state will occur and produce a negative value at node B [9].

## 2.3. Tunneling Phase Logic Technology

A tunneling phase logic (TPL) minority gate is the basic unit used in TPL technology to implement logic circuits. As shown in **Figure 4**, the inputs of a TPL minority gate are three waveforms ($W_1$, $W_2$, and $W_3$). The phases of a waveform are used to represent logic 0 and 1. Based on the input waveforms, the phase of the output waveform is determined. If two phases of the three input waveforms

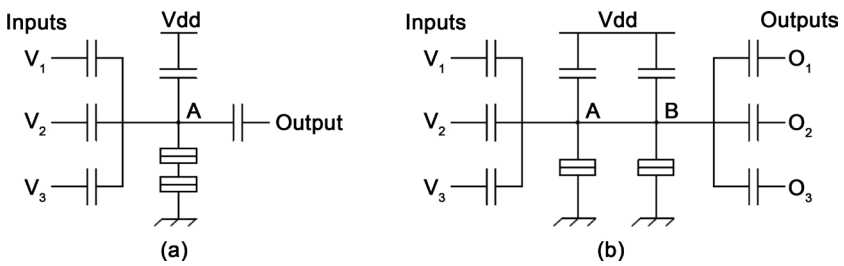


**Figure 3.** (a) SET minority gate; (b) SET majority gate.

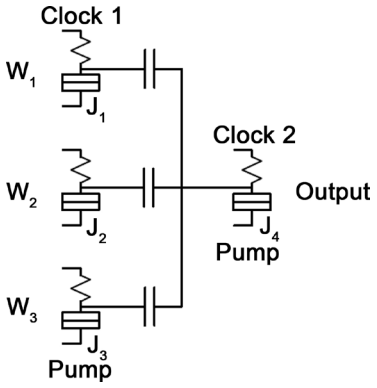


**Figure 4.** TPL minority gate.

are different, they will neutralize each other and the reverse of the third waveform will be the output. However, if all input waveforms have the same phases, the output will be the reverse of these phases.

### 2.4. Spintronic Majority Gate

A spintronic majority gate (SMG) is a device that performs a three-input majority function. This device is implemented with a cross of ferromagnetic wires with a size of 140 × 140 nm [29] [30]. Over the four ends of the cores, the three inputs (A, B, C) and output (Out) terminals are formed as nanopillars (20 × 20 nm each) with a separate ferromagnetic layer as shown in **Figure 5(a)** and **Figure 5(b)**. Based on the polarity of the input voltage (either positive (+V) or negative (−V)) applied on each nanopillar, the current exerts spin torque in order to switch the magnetization of the common layer to a certain direction. The final direction of the magnetization (Iout) is determined by the majority directions of the inputs and sensed via the tunneling magnetoresistance (TMR) effect using a sense amplifier. **Figure 5(b)** shows the width and length of arms, the size of pillars, and distance between them (a = 20 nm).

### 2.5. All Spin Logic

An all spin logic (ASL) device is also spin based device [31] [32]. It constructed of copper wires and nanomagnets. To implement an ASL device that performs a three-input majority function, four nanomagnets, which represent the three inputs and one output, are placed over the ends of the copper wires as shown in **Figure 6(a)**. The input and output sides of each of these nanomagnets are separated by an insulator. Due to the current driven to the ground terminal from the voltage supplied to the top of each nanomagnets, spin polarized electrons accumulate in the two sides of each nanomagnet with different concentrations. This difference causes a diffusion spin current, which exerts torque on a nanomagnet and is able to switch its polarization. Based on the majority of input polarizations, the output is determined and delivered via the output nanomagnet as a logic value. The inverters also can be implemented based on the same properties of polarization changes as shown in **Figure 6(b)**.

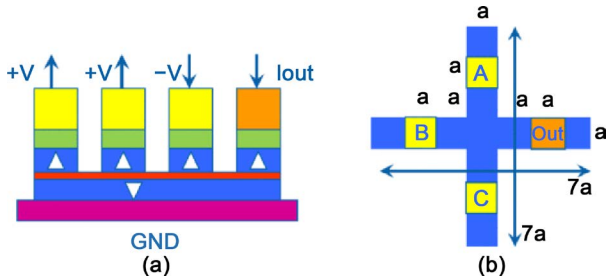


Figure 5. (a) SMG device [29]; (b) Top view of SMG [29].

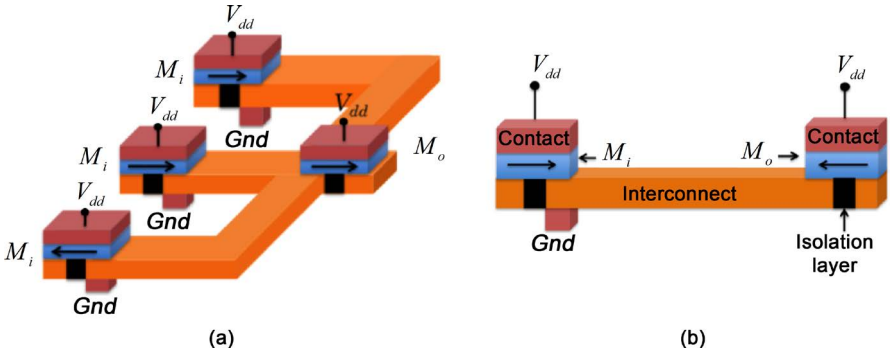


Figure 6. ASL devices: (a) ASL majority gate [33]; (b) ASL inverter [33].

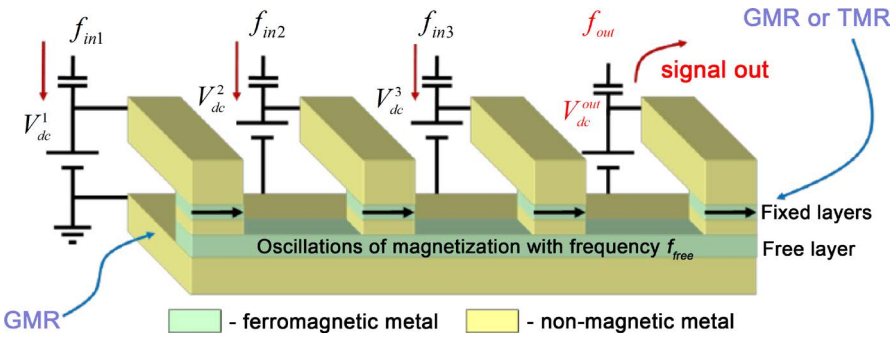


Figure 7. STO logic majority gate [11].

## 2.6. Spin Torque Oscillator Logic

A spin torque oscillator (STO) logic is a device that can perform a three-input majority function [34]. This device consists of four nanopillars (three inputs and one output) with their own layers. Similar to SMG, the oscillators have a common ferromagnetic layer as shown in Figure 7. The input currents pass through nanopillars and exert spin torques that drive oscillators. Because of the driven oscillators, spin waves propagate in the common layer that make the oscillators signal coupled. Based on the majority of the inputs, the frequency of the output oscillator is determined. This can be sensed via the effect of giant magnetoresistance (GMR) or TMR. The frequency of the output serves in the circuit as the logic signal.

## 2.7. Spin Wave Device Technology

In spin wave device (SWD) technology, computation and information transfor-

mation occur via spin waves [11] [35] [36]. SWD technology uses the majority gate as the logic primitive. A SWD majority gate is constructed of the symmetric merging of three waveguides [37] [38]. Its operation is based on the interference of the input spin waves. The output is determined based on the interference of the three phases of the input spin waves via magneto-electron (ME) cells [39]. Another logic device in SWD technology is a SWD inverter which is implemented by a waveguide to deliver the inverse of spin wave signal to the output ME cell [38] [40]. **Figure 8(a)** and **Figure 8(b)** show the areas and designs for a SWD majority gate and inverter, respectively. It can be seen that the length of waveguide in the inverter is 1.5× of the length of spin wavelength ($\lambda_{SW}$), while the length of each waveguide in the majority gate is 1.0× of the spin wavelength.

## 2.8. Nanomagnetic Logic

The process of computation and information transformation in nanomagnetic logic (NML) [42] is based on magnetization of patterned array of elongated nanomagnets. In NML, there are two stable magnetization states of magnets that are used to represent binary information. These states are commonly referred to as "up" or "down" which represent logic 1 or 0, respectively, as shown in **Figure 9(a)**. The fundamental logic element in NML technology is a three-input majority gate. This gate is constructed of a cross of five dots, which are one central dot surrounded by four dots that represent inputs (A, B, C) and output as shown in **Figure 9(b)**. Based on the majority of magnetization of the three inputs, the output is calculated via magnetic interactions.

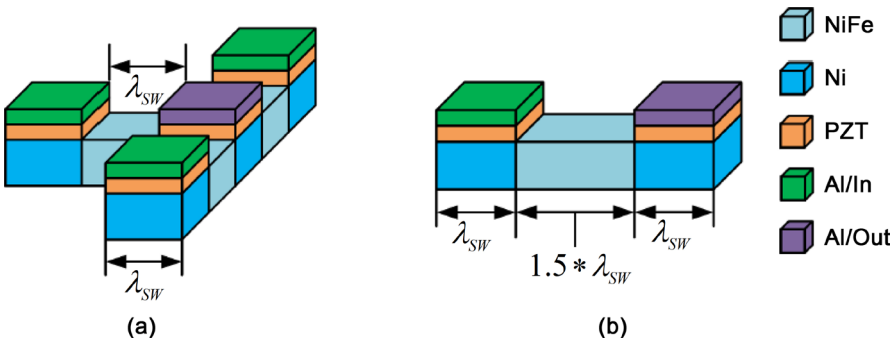


**Figure 8.** SWD devices: (a) SWD majority gate [41]; (b) SWD inverter [41].

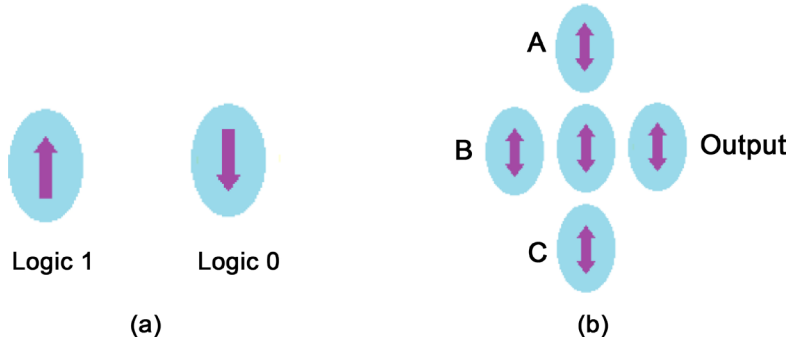


**Figure 9.** (a) The possible stable magnetizations; (b) NML majority gate.

## 2.9. DNA Technology

DNA technology is being considered as a possible alternative to silicon-based technologies especially for implantable medical devices. The small size, light weight, and compatibility with bio-signals of DNA technology show its ability of implementing logic circuits. Several researchers have introduced different designs of DNA majority gates [43] [44] [45] based on different techniques such as the four-way junction-driven DNA majority gate, spatially localised DNA majority gate, etc. The basic operation associated with such majority gates is the DNA strand displacement mechanism. Different designs of DNA majority gates are shown in Figure 10.

In addition to the nanotechnologies discussed in this paper, other nanotechnologies such as graphene [46] [47] reconfigurable gate [48], resistive RAM [49] [50] [51], carbon nanotube [52] [53] [54] [55], etc., use logic majority and/or minority gates as circuit primitives. Hence, in order to implement an efficient logic circuit in any of these nanotechnologies including all the nanotechnologies

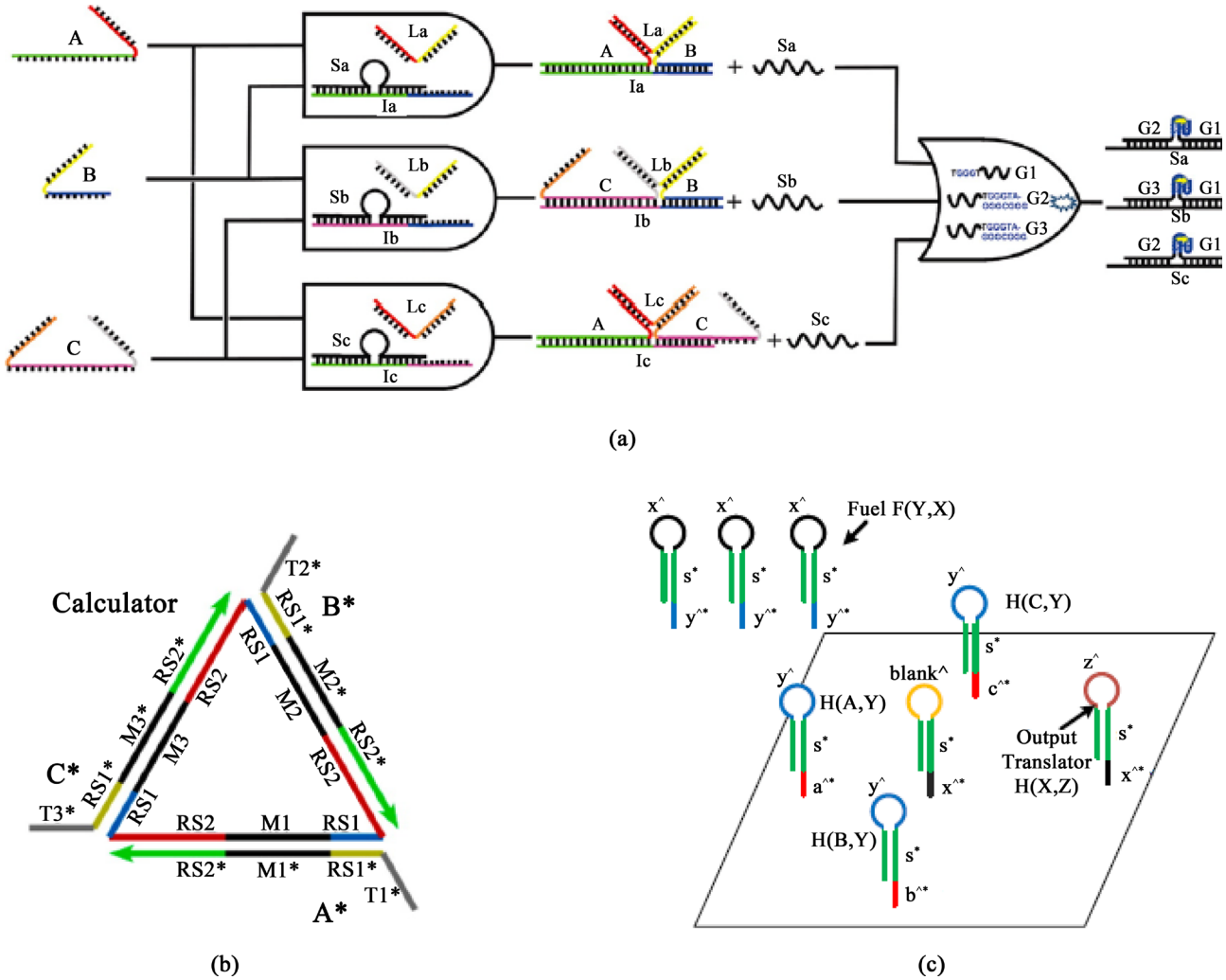


Figure 10. Different designs of DNA majority gates: (a) Four-way junction-driven DNA majority gate [43]; (b) DNA majority gate given in [44]; (c) Spatially localised DNA majority gate [45]. (Each color represents a particular domain in the strand).

discussed earlier, the circuit has to be converted into its equivalent majority- or minority-based logic circuit.

As mentioned earlier, since minority logic is the complement of majority function, De Morgan's theorem can be used to drive a minority logic network from its equivalent majority network. This process results in a minority network with the same number of majority gates and levels as in its equivalent majority network. This means that an efficient majority network synthesis method can be used to obtain both majority and minority networks. The simplified Boolean functions expressed in standard forms SOP and POS can be directly converted into majority or minority logic networks by implementing the majority AND/OR mapping method. This method is to map each logic gate in the simplified Boolean functions to majority AND/OR gates. However, in most cases, this method does not results in optimal majority/minority expressions. In other words, the number of gates, levels, etc., used in majority/minority expressions obtained from the AND/OR mapping method are not the optimal results. For example, consider the majority function $f = x_1x_2 + x_1x_3 + x_2x_3$. By using the AND/OR mapping method, it requires five majority gates, three levels as $n_1 = x_1x_2$, $n_2 = x_1x_3$, $n_3 = x_2x_3$, $n_4 = n_1 + n_2$, and $f = n_3 + n_4$, whereas it can be realized with only one majority gate in one level, *i.e.*, $f = M(x_1, x_2, x_3)$. Therefore, an efficient majority/minority logic network synthesis is needed in order to generate optimal majority/minority logic networks. In the next section, we review the best existing comprehensive majority/minority logic synthesis methods in detail.

## 3. Majority/Minority Logic Synthesis Methods

Several researchers have proposed different techniques for majority/minority logic synthesis. However, none of these techniques are capable of generating optimal majority or minority expressions in terms of gates, levels, inverters and gate inputs for all cases. In addition, only a few of these methods can be used for synthesizing multi-input multi-output majority/minority logic networks. These methods are discussed in detail as follows:

### 3.1. Majority Logic Synthesis (MALS) [20]

MALS is the first proposed comprehensive majority/minority logic network synthesis method that is capable of synthesizing multi-level multi-output majority/minority logic networks. The input to MALS is a minimized algebraically factored multi-output combinational network, and the output is an equivalent majority logic network. The method starts by preprocessing and decomposing the input network such that each node in the decomposed network has at most three input variables. This process is done by using preprocessing and decomposition methods in SIS. The next step is to check each decomposed node to see whether it is a majority function. If so, the node will be converted and the process will move to check the next node. Otherwise, the node function will be checked if there is any common literal. If this is the case, the literal will be factored out and

an AND/OR mapping is then performed on the factored function. If the node function has no common literal and it can be realized with less than four majority AND/OR gates, an AND/OR mapping will then be performed. Otherwise, the node will be converted into its equivalent majority expression with at most four majority gates in two levels using K-map. This procedure is accomplished by first getting the K-map of the logic function of the node. Next, the first majority function $f_1$ is determined by finding the admissible pattern from the K-map of the node. Based on the K-map of the node and the first admissible pattern, the second admissible pattern is then found which gives the second majority function $f_2$. Lastly, the third admissible pattern is found based on the K-map of the node and the first and the second admissible patterns. The third admissible pattern gives the third majority function $f_3$. These three majority functions are determined such that the original node can be replaced with the majority function of these three functions as $M(f_1, f_2, f_3)$.

### 3.2. Kong's Synthesis [21]

Another comprehensive majority/minority logic network synthesis method was introduced by Kong *et al.* [21]. The input to this methodology is an arbitrary multi-output Boolean function, and its output is an equivalent majority logic network. The method begins by preprocessing the input network and checking its correctness using SIS. If the input function is correct, multiple preprocessing scripts given in **Figure 11** are applied to simplify and factor it algebraically, where all "(*x*)" are replaced with "3". Otherwise, error information will be shown and the process will be ended. After preprocessing, the factored functions are decomposed using SIS such that each node has at most three input variables. For decomposition, four different methods given in **Figure 12** are performed in order to obtain the minimum number of three-feasible nodes. In these decomposition methods, all "(*x*)" are replaced with "3" to produce three-feasible decomposed networks. After decomposition, all nodes in the decomposed network are then checked to see if there is any node that can be collapsed into its fanout while retaining feasibility. This process can reduce further the number of nodes. In the next step, each node in the decomposed network is then checked to see if it is a majority function. If so, the function is then converted into its corresponding majority expression based on forty primitive functions which are all the possible three-variable Boolean functions. Otherwise, all admissible expression groups are found from the forty majority expressions such that each group consists of three majority expressions $(f_1, f_2, f_3)$ where the node is the majority function of these expressions, *i.e.*, $M(f_1, f_2, f_3)$. Then, all the majority functions that consist of expression groups with a minimum number of majority gates are selected. Next, the selected functions are checked to select the functions with a minimum number of gate inputs. Then, the selected functions are checked again to select the functions with a minimum number of inverters. The same steps are repeated for the complement of the node function. The next step

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

**Figure 11.** Preprocessing script used in [21] and [26].

**Method 1:** *xl_split − n* (*x*)
**Method 2:** *xl_imp − n* (*x*)
**Method 3:** *xl_part_coll − n* (*x*) *− m − g* 2
*xl_coll_ck − n* (*x*)
*xl_partition − n* (*x*) *− m*
*full_simplify*
*xl_imp − n* (*x*)
*xl_partition − n* (*x*) *− t*
*xl_cover − n* (*x*) *− e* (*x*)0 *− u* 200
*xl_coll_ck − n* (*x*) *− k*
**Method 4:** *xl_part_coll − n* (*x*) *− m − g* 2
*xl_coll_ck − n* (*x*)
*xl_partition − n* (*x*) *− m*
*sweep; eliminate* − 1
*simplify − m nocomp*
*eliminate* − 1
*sweep; eliminate* 5
*simplify − m nocomp*
*resub − a*
*fx*
*resub − a; sweep*
*eliminate* − 1*; sweep*
*full_simplify − m nocomp*
*xl_imp − n* (*x*)
*xl_partition − n* (*x*) *− t*
*xl_cover − n* (*x*) *− e* (*x*)0 *− u* 200
*xl_coll_ck − n* (*x*) *− k*

**Figure 12.** Four decomposition methods scripts used in [21] and [26].

is to select the majority function with a minimum number of majority gates, gate inputs, and inverters from the selected majority functions that consist of expression groups and their complements. The last step is to check obtained majority expressions and see if there are repeated nodes. If so, these nodes will be removed and the majority network will be updated. This process keeps running until no repeated nodes exist.

### 3.3. Majority Expression Lookup Table (MLUT)-Based Synthesis [26]

One of the majority/minority logic network synthesis methods is MLUT-based method [26]. The input to this method is an arbitrary Boolean functions network, and the output is an equivalent majority logic network. This method also starts by preprocessing and decomposing the input network using SIS as used in Kong's method. However, the preprocessing and decomposition methods used here are able to preprocess and decompose the input Boolean functions network up to four-feasible networks. In preprocessing, the input Boolean functions are simplified by algebraically factoring the common terms out and removing the repeated terms by applying the preprocessing script given in **Figure 11**, where all "($x$)" are replaced with "4". For decomposition, the same four methods used in Kong's method are implemented in order to find the minimum number of decomposed networks. However, these four decomposition methods will decompose the network into two-feasible, three-feasible and four-feasible networks by replacing all "($x$)" in **Figure 12** with "2", "3", and "4" in order to find the best solution. In this method, a majority expression lookup table is developed. This table is developed by generating equivalent majority expressions for all possible four-variable Boolean functions. This results in a table that contains ninety four-variable Boolean functions and their equivalent majority expressions. Based on this table, each decomposed node, which consists of four or fewer variables, is then converted into its corresponding majority expression. A redundancy removal method is also used. This process can provide further simplification by implementing several steps. It starts by checking all nodes in the obtained majority network and removing the repeated nodes. All nodes with duplicated inputs are then simplified. The next step is to sweep all nodes without majority gates. The last step in the redundancy removal method is to minimize the number of inverters. This step is implemented if the majority network has two cascaded inverters which can cancel each other out. Another case would be if a majority gate in the network has two or three internal inverters that can be factored out to have only one external inverter. The redundancy removal method may require more than one iteration until no further simplification is possible.

## 4. Comparison and Discussion

### 4.1. Comparison of the Comprehensive Synthesis Methods

The three majority/minority synthesis methods discussed in this paper differ from each other in their preprocessing methods, decomposition methods, conversion techniques, and optimization targets: gates, levels, inverters, and gate inputs. **Table 1** gives a summary of these differences. These differences are discussed in detail as follows:

#### 4.1.1. Preprocessing

The first step in all three synthesis methods is preprocessing. This process is

**Table 1.** Comparison of the best comprehensive synthesis methods.

| Method | Preprocessing ($r$-feasible networks) | Decomposition | | | Conversion technique | Optimization targets | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | Methods | $r$-feasible networks | Node reduction | | Gates | Levels | Inverters | Gate inputs |
| MALS [20] | $r=3$ | 1 | $r=3$ | No | K-map | Yes | No | No | No |
| Kong's [21] | $r=3$ | 1, 2, 3, 4 | $r=3$ | Yes | 40 Primitive functions | Yes | No | Yes | Yes |
| MLUT [26] | $r=4$ | 1, 2, 3, 4 | $r=2,3,4$ | Yes | 90 Primitive functions | Yes | Yes | Yes | Yes |

used to simplify the input Boolean functions by removing the redundant terms and algebraically factoring the common terms out. For example, consider the Boolean function $F = x_1x_2' + x_1x_3 + x_2'x_3 + x_1x_2'x_3$. This function is first simplified to $F = x_1x_2' + x_1x_3 + x_2'x_3$. Then, the common terms are factored out and the function is simplified to $F = (x_2' + x_3)x_1 + x_2'x_3$. In all algorithms, this process is done by using the simplification and factorization methods in SIS. However, the preprocessing method used in MLUT is improved by performing the operations of kernel and cube extraction for four-feasible networks instead of three-feasible networks as used in MALS and Kong's method.

Although, the preprocessing method provides simplified Boolean functions in terms of logic AND, OR and NOT, these functions are not expressed properly to be converted into their equivalent majority expressions for some cases. To demonstrate this point, consider the same function that we used for simplification. After removing the redundant term, the function is expressed by $F = x_1x_2' + x_1x_3 + x_2'x_3$. It can be seen that this function is expressed as a majority function which can be realized with only one majority gate in one level, *i.e.*, $F = M(x_1, x_2', x_3)$. However, if the common terms are algebraically factored out, *i.e.*, $F = (x_2' + x_3)x_1 + x_2'x_3$, the function will have a different expression which can result in an equivalent majority expression with more than one majority gate and one level. This specific example may not fall in this category due to its simplicity. However, this case can occur especially while processing large circuits which can cause a large change in the final result.

#### 4.1.2. Decomposition

For decomposition, MALS uses method 1 in Figure 12 to decompose the input network into smaller nodes such that each node in the network has at most three input variables which can be easily converted into its equivalent majority expression. In Kong's method, the decomposition process is also used to decompose the input networks into three-feasible networks. However, this method uses four different decomposition methods as given in Figure 12. Any function with three input variables can be realized with at most four majority gates in two levels [14] [56]. Thus, the total number of majority gates in the synthesized majority network is between the number of nodes and the number of nodes multiplied by 4. Therefore, in order to reduce the number of majority gates in a synthesized majority network, the number of nodes must be reduced. None of the four decomposition methods used in Kong's method give the minimum number of nodes

for all cases. Therefore, all four methods are applied to find out the best results. In MLUT, the same four decomposition methods are used. However, these methods are improved to decompose the input network into two-feasible, three-feasible, and four-feasible networks. Based on the obtained networks from the four decomposition methods, the best solution is then chosen. The obtained decomposed networks from the four methods are not guaranteed to be optimal. However, they provide a fundamental library of heuristic techniques for decomposition. In both Kong's method and MLUT, all nodes in the obtained decomposed networks are then checked to see if there are nodes that can be collapsed into their fanouts while retaining feasibility. This can provide further reduction in the number of gates, levels, inverters, and gate inputs. However, this process is not considered in MALS.

#### 4.1.3. Converting Boolean Functions into Majority Expressions

For converting the decomposed networks into their equivalent majority expressions, each method uses a different technique. The MALS method uses K-map to obtain one-level majority functions $f_1, f_2$ and $f_3$ for each node, such that the function can be represented as $M(f_1, f_2, f_3)$. This method can generate only one admissible majority expression for a given Boolean function. This is considered as a drawback for this method. Therefore, this technique does not guarantee that it results in optimal majority expressions. In Kong's method, the process of converting the function of a node is based on forty optimal majority expressions. If the Boolean function belongs to these forty expressions, it is converted into its corresponding majority expression. Otherwise, all admissible three-expression groups from the forty expressions are found such that the function of the node can be represented as a majority function of the three expressions. This conversion technique is also used in MLUT. However, this method is based on ninety primitive functions instead of forty as used in Kong's method. These primitives are the equivalent majority expressions for all possible four-variable Boolean functions. Each node in the decomposed network is replaced with its equivalent majority expressions if it has a corresponding expression. Otherwise, a combination of three majority expressions is chosen from the ninety expressions such that the function of the node can be represented as the majority function of the chosen three expressions.

#### 4.1.4. Optimization Targets

Since the gate count and level count determine the latency and the size of a majority/minority circuit, they are the most important factors that play an essential role in enhancing performance. Therefore, by reducing the number of gates and the number of levels, the performance can be improved. In the three comprehensive synthesis methods (MALS, Kong's, and MLUT), the optimization is targeted to reduce either the number of gates or levels. In MALS and Kong's method, the gate count reduction is taken as the first priority for optimization. However, in MLUT, either the number of gates or levels can be taken as the first

priority. In addition to the number of gates and level count, there are other factors that can play an essential role in providing further scaling down of feature sizes of a generated majority circuit. One of these factors is inverter count. In some nanotechnologies, the implementation of an inverter requires a larger area than a majority gate. For example, in QCA technology, the implementation of an inverter requires seven QCA cells as shown in **Figure 2(b)**, whereas five QCA cells are required to implement a majority gate as shown in **Figure 2(c)**. Another factor that can lead to additional optimization is the number of gate inputs. For example, consider the majority functions $F_1 = M\left(x_3, 1, M\left(x_1, x_2, x_3\right)\right)$ and $F_2 = M\left(x_3, 1, M\left(x_1, x_2, 0\right)\right)$. Both $F_1$ and $F_2$ are the equivalent majority expressions to the Boolean function $F = x_1x_2 + x_3$. However, $F_1$ has four gate inputs and $F_2$ has three gate inputs. Since the logic 0 and 1 in QCA technology can be generated from external sources at their positions, the best majority expression to implement this circuit in QCA is $F_2$. Therefore, by reducing the number of gate inputs of a circuit, the routing complexity can be reduced. The optimization of inverters and gate inputs are only considered in Kong's method and MLUT.

### 4.2. Comparison of Experimental Results

In this section, we demonstrate an overall comparison between the results obtained from the existing synthesis methods. In **Table 2**, the obtained equivalent majority expressions for eight standard three-variable Boolean functions [15] using the comprehensive synthesis methods discussed in this paper and other five three-variable synthesis methods [15] [16] [17] [18] [19] are given. In the same table, the numbers of majority gates, levels, inverters, and gate inputs used in each majority expression are given as well. From the table, it can be seen that MALS results in an optimal solution for some functions, whereas Kong's method and MLUT give the optimal expressions in terms of gates, levels and inverters for all Boolean functions. However, none of these methods result in the minimum number of gate inputs for all functions. For example, the equivalent majority expression for the Boolean function $F = x_1x_2 + x_1'x_2'x_3$ obtained from Kong's method and MLUT is $F = M\left(M\left(x_1, 0, x_2\right), M\left(x_1, 1, x_2\right)', M\left(x_1, 1, x_3\right)\right)$. This expression requires four gates, two levels, one inverter and nine gate inputs. From the AND/OR mapping method and methods in [15] [18] [20], the obtained majority expression is $F = M\left(M\left(M\left(x_1', 0, x_2'\right), 0, x_3\right), 1, M\left(x_1, 0, x_2\right)\right)$. This expression requires four gates, two levels, one inverter and eight gate inputs.

For Boolean functions with more than three variables, we compare the results of 40 Microelectronics Center North Carolina benchmark circuits [57] using the comprehensive synthesis methods in **Table 3**. The results obtained from each of the three methods are compared with the majority AND/OR mapping method. In this table, only the number of majority gates and levels are considered for comparison. As shown in the table, when the MLUT method is targeted to reduce the number of gates, there is an average reduction of 36.1% in gate counts

**Table 2.** Comparison of 8 three-variable standard Boolean functions using exist synthesis methods.

| Standard function | Method | Majority expression | Gates | Levels | Inverters | Gate inputs |
|---|---|---|---|---|---|---|
| $F = x_1x_2 + x_1'x_2'x_3$ | AND/OR mapping | $M\left(M\left(M\left(x_1',0,x_2'\right),0,x_3\right),1,M\left(x_1,0,x_2\right)\right)$ | 4 | 3 | 2 | 8 |
| | [15] [18] [20] | $M\left(M\left(M\left(x_1',0,x_2'\right),0,x_3\right),1,M\left(x_1,0,x_2\right)\right)$ | 4 | 3 | 2 | 8 |
| | [16] | $M\left(M\left(x_1',1,x_2\right),M\left(x_1,0,x_2\right),M\left(x_2',0,x_3\right)\right)$ | 4 | 2 | 2 | 9 |
| | [17] | $M\left(M\left(x_1,0,x_2\right),M\left(x_1,1,x_2\right)',M\left(x_1,1,x_3\right)\right)$ | 4 | 2 | 1 | 9 |
| | [19] | $M\left(M\left(x_1',1,x_2\right),M\left(x_1,x_2',x_3\right),M\left(x_1,0,x_2\right)\right)$ | 4 | 2 | 2 | 10 |
| | [21] [26] | $M\left(M\left(x_1,0,x_2\right),M\left(x_1,1,x_2\right)',M\left(x_1,1,x_3\right)\right)$ | 4 | 2 | 1 | 9 |
| $F = x_1x_2x_3 + x_2x_3' + x_1x_2'x_3'$ | AND/OR mapping | $M\left(M\left(M\left(M\left(x_1,0,x_2\right),0,x_3\right),0,M\left(M\left(x_1',0,x_2\right),0,x_3'\right)\right),\right.$ $\left.0,M\left(M\left(x_1,0,x_2'\right),0,x_3'\right)\right)$ | 8 | 4 | 4 | 16 |
| | [15] [16] | $M\left(M\left(x_1,0,x_3\right),M\left(x_1,x_2,x_3'\right),M\left(x_1',x_2',x_3'\right)\right)$ | 4 | 2 | 4 | 11 |
| | [17] | $M\left(M\left(M\left(x_1,x_2,x_3\right),1,x_1\right)',x_3,M\left(x_1,x_2,x_3'\right)\right)$ | 4 | 3 | 2 | 11 |
| | [18] [20] | $M\left(M\left(x_1,x_2,x_3'\right),M\left(x_1',0,x_3'\right),M\left(x_1,x_2',x_3\right)\right)$ | 4 | 2 | 4 | 11 |
| | [19] | $M\left(M\left(x_1,x_2,x_3'\right),M\left(x_1,x_2',x_3\right),M\left(x_1',0,x_2\right)\right)$ | 4 | 2 | 3 | 11 |
| | [21] [26] | $M\left(M\left(x_1,0,x_3\right),M\left(x_1,x_2,x_3\right)',M\left(x_1,x_2,x_3'\right)\right)$ | 4 | 2 | 2 | 11 |
| $F = x_1 + x_2x_3$ | AND/OR mapping | $M\left(M\left(x_2,0,x_3\right),1,x_1\right)$ | 2 | 2 | 0 | 4 |
| | [15] [16] [18] [19] | $M\left(M\left(x_2,0,x_3\right),1,x_1\right)$ | 2 | 2 | 0 | 4 |
| | [17] | $M\left(M\left(x_1,1,x_3\right),x_1,x_2\right)$ | 2 | 2 | 0 | 5 |
| | [20] [21] [26] | $M\left(M\left(x_2,0,x_3\right),1,x_1\right)$ | 2 | 2 | 0 | 4 |
| $F = x_1x_2 + x_2'x_3$ | AND/OR mapping | $M\left(M\left(x_1,0,x_2\right),1,M\left(x_2',0,x_3\right)\right)$ | 3 | 2 | 1 | 6 |
| | [15] [16] [18] [19] | $M\left(M\left(x_1,0,x_2\right),1,M\left(x_2',0,x_3\right)\right)$ | 3 | 2 | 1 | 6 |
| | [17] | $M\left(M\left(x_1,1,x_2\right)',x_1,M\left(x_2,1,x_3\right)\right)$ | 3 | 2 | 1 | 7 |
| | [20] [21] [26] | $M\left(M\left(x_1,0,x_2\right),1,M\left(x_2',0,x_3\right)\right)$ | 3 | 2 | 1 | 6 |
| $F = x_1x_2x_3 + x_1'x_2'x_3'$ | AND/OR mapping | $M\left(M\left(M\left(x_1,0,x_2\right),0,x_3\right),1,M\left(M\left(x_1',0,x_2'\right),0,x'\right)\right)$ | 5 | 3 | 3 | 10 |
| | [15] [18] [20] | $M\left(M\left(M\left(x_1,0,x_2\right),0,x_3\right),1,M\left(M\left(x_1',0,x_2'\right),0,x'\right)\right)$ | 5 | 3 | 3 | 10 |
| | [16] [19] | $M\left(M\left(x_1',1,x_2\right),M\left(x_2',0,x_3'\right),M\left(x_1,0,x_3\right)\right)$ | 4 | 2 | 3 | 9 |
| | [17] | $M\left(M\left(x_1,1,x_2\right)',M\left(x_2,0,x_3\right),M\left(x_1',0,x_3'\right)\right)$ | 4 | 2 | 2 | 9 |
| | [21] [26] | $M\left(M\left(x_1,1,x_2\right),M\left(x_2,1,x_3\right)',M\left(x_1',1,x_3\right)\right)$ | 4 | 2 | 2 | 9 |

**Continued**

| | | | | | | |
|---|---|---|---|---|---|---|
| $F = x_1x_2 + x_2x_3 + x_1'x_2'x_3'$ | AND/OR mapping | $M\left(M\left(M\left(x_1,0,x_2\right),1,M\left(x_2,0,x_3\right)\right),1,M\left(M\left(x_1',0,x_2'\right),0,x_3'\right)\right)$ | 6 | 3 | 3 | 12 |
| | [15] [16] [19] | $M\left(M\left(M\left(x_1,1,x_3\right),0,x_2\right),1,M\left(M\left(x_2',0,x_3'\right),0,x_1'\right)\right)$ | 5 | 3 | 3 | 10 |
| | [17] | $M\left(M\left(x_1,x_2,x_3\right),M\left(x_1',1,x_2'\right),M\left(x_2',0,x_3'\right)\right)$ | 4 | 2 | 3 | 10 |
| | [18] [20] | $M\left(M\left(x_1,x_2,x_3\right),M\left(x_1',0,x_2'\right),M\left(x_2',1,x_3'\right)\right)$ | 4 | 2 | 3 | 10 |
| | [21] [26] | $M\left(M\left(x_1',1,x_2\right),M\left(x_2,1,x_3\right)',M\left(x_1,x_2,x_3\right)\right)$ | 4 | 2 | 2 | 10 |
| $F = x_1x_2x_3 + x_1'x_2'x_3 + x_1x_2'x_3' + x_1'x_2x_3'$ | AND/OR mapping | $M\left(M\left(M\left(M\left(x_1,0,x_2\right),0,x_3\right),1,M\left(M\left(x_1',0,x_2'\right),0,x_3\right)\right),1,M\left(M\left(M\left(x_1,0,x_2'\right),0,x_3'\right),1,M\left(M\left(x_1',0,x_2\right),0,x_3'\right)\right)\right)$ | 11 | 5 | 6 | 22 |
| | [15] [16] [19] | $M\left(M\left(x_1',x_2,x_3\right),x_3',M\left(x_1,x_2',x_3\right)\right)$ | 3 | 2 | 3 | 9 |
| | [17] | $M\left(M\left(x_1,x_2,x_3\right)',x_3,M\left(x_1,x_2,x_3'\right)\right)$ | 3 | 2 | 2 | 9 |
| | [18] [20] | $M\left(M\left(x_1',x_2,x_3\right),M\left(x_1,x_2,x_3'\right),M\left(x_1,x_2',x_3\right)\right)$ | 4 | 2 | 3 | 12 |
| | [21] [26] | $M\left(M\left(x_1,x_2,x_3\right)',x_1,M\left(x_1',x_2,x_3\right)\right)$ | 3 | 2 | 2 | 9 |
| $F = x_1x_2x_3 + x_1x_2'x_3'$ | AND/OR mapping | $M\left(M\left(M\left(x_1,0,x_2\right),0,x_3\right),1,M\left(M\left(x_1,0,x_2'\right),0,x_3'\right)\right)$ | 5 | 3 | 2 | 10 |
| | [15] [16] [19] | $M\left(M\left(x_1,x_2,x_3'\right),0,M\left(x_1,x_2',x_3\right)\right)$ | 3 | 2 | 2 | 8 |
| | [17] | $M\left(M\left(x_2,0,x_3\right),x_1,M\left(x_2,1,x_3\right)'\right)$ | 3 | 2 | 1 | 7 |
| | [18] [20] | $M\left(M\left(M\left(x_2',0,x_3'\right),1,M\left(x_2,0,x_3\right)\right),0,x_1\right)$ | 4 | 3 | 2 | 8 |
| | [21] [26] | $M\left(M\left(x_2,0,x_3\right),x_1,M\left(x_2,1,x_3\right)'\right)$ | 3 | 2 | 1 | 7 |

**Table 3.** Comparison of 40 Benchmarks using the best exist synthesis methods.

| Benchmark | AND/OR mapping | | MALS [20] | | Kong's [21] | | MLUT [26] | | | | Reduction% | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | Gate priority | | Level priority | | MALS [20] | | Kong's [21] | | MLUT [26] (Gates) | | MLUT [26] (Levels) | |
| | Gates | Levels | Gates | Levels | Gates | Levels | Gates | Levels | Levels | Gates | Gates | Levels | Gates | Levels | Gates | Levels | Levels | Gates |
| b1 | 9 | 3 | 9 | 3 | 7 | 2 | 6 | 2 | 2 | 6 | 0.0 | 0.0 | 22.2 | 33.3 | 33.3 | 33.3 | 33.3 | 33.3 |
| cm42a | 21 | 2 | 21 | 2 | 18 | 2 | 18 | 2 | 2 | 18 | 0.0 | 0.0 | 14.3 | 0.0 | 14.3 | 0.0 | 0.0 | 14.3 |
| decod | 28 | 3 | 28 | 3 | 28 | 3 | 28 | 3 | 3 | 28 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| cm82a | 50 | 7 | 16 | 8 | 7 | 3 | 6 | 3 | 3 | 6 | 68.0 | −14.3 | 86.0 | 57.1 | 88.0 | 57.1 | 57.1 | 88.0 |
| majority | 12 | 5 | 6 | 5 | 6 | 4 | 5 | 4 | 3 | 6 | 50.0 | 0.0 | 50.0 | 20.0 | 58.3 | 20.0 | 40.0 | 50.0 |
| z4ml | 71 | 10 | 27 | 8 | 9 | 4 | 9 | 4 | 4 | 9 | 62.0 | 20.0 | 87.3 | 60.0 | 87.3 | 60.0 | 60.0 | 87.3 |
| 9symml | 276 | 15 | 216 | 12 | 47 | 10 | 45 | 12 | 10 | 47 | 21.7 | 20.0 | 83.0 | 33.3 | 84.0 | 20.0 | 33.3 | 73.2 |
| ldd | 91 | 9 | 73 | 13 | 67 | 7 | 67 | 7 | 7 | 67 | 19.8 | −44.4 | 26.4 | 22.2 | 26.4 | 22.2 | 22.2 | 26.4 |
| alu2 | 495 | 15 | 354 | 18 | 340 | 18 | 329 | 18 | 16 | 347 | 28.5 | −20.0 | 31.3 | −20.0 | 33.5 | −20.0 | −6.7 | 29.9 |

**Continued**

| | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| x2 | 49 | 6 | 42 | 8 | 37 | 7 | 34 | 7 | 6 | 36 | 14.3 | −33.3 | 24.5 | −16.7 | 30.6 | −16.6 | 0.0 | 26.5 |
| cm152a | 31 | 5 | 21 | 5 | 21 | 6 | 17 | 6 | 6 | 17 | 32.3 | 0.0 | 32.3 | −20.0 | 45.2 | −20.0 | −20.0 | 45.2 |
| cm85a | 80 | 10 | 34 | 10 | 26 | 6 | 19 | 6 | 6 | 19 | 57.5 | 0.0 | 67.5 | 40.0 | 76.3 | 40.0 | 40.0 | 76.3 |
| cm151a | 56 | 8 | 42 | 8 | 23 | 7 | 20 | 7 | 7 | 20 | 25.0 | 0.0 | 58.9 | 12.5 | 64.3 | 12.5 | 12.5 | 64.3 |
| cm162a | 57 | 7 | 46 | 9 | 41 | 7 | 36 | 9 | 7 | 41 | 19.3 | −28.6 | 28.1 | 0.0 | 36.8 | -28.6 | 0.0 | 28.1 |
| cu | 61 | 8 | 46 | 7 | 40 | 7 | 39 | 7 | 6 | 40 | 24.6 | 12.5 | 34.4 | 12.5 | 36.1 | 12.5 | 25.0 | 34.4 |
| cm163a | 52 | 7 | 42 | 9 | 38 | 7 | 32 | 7 | 7 | 32 | 19.2 | −28.6 | 26.9 | 0.0 | 38.5 | 0.0 | 0.0 | 38.5 |
| cmb | 44 | 4 | 44 | 5 | 28 | 4 | 26 | 4 | 4 | 26 | 0.0 | −25.0 | 36.4 | 0.0 | 40.1 | 0.0 | 0.0 | 40.1 |
| pm1 | 49 | 6 | 45 | 7 | 35 | 6 | 32 | 6 | 6 | 32 | 8.2 | −16.7 | 28.6 | 0.0 | 34.7 | 0.0 | 0.0 | 34.7 |
| tcon | 24 | 2 | 24 | 2 | 24 | 2 | 24 | 2 | 2 | 24 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| vda | 856 | 13 | 738 | 14 | 700 | 15 | 670 | 14 | 14 | 670 | 13.8 | −7.7 | 18.2 | −15.4 | 21.7 | −7.7 | −7.7 | 21.7 |
| pcle | 78 | 9 | 67 | 8 | 62 | 8 | 62 | 8 | 8 | 62 | 14.1 | 11.1 | 20.5 | 11.1 | 20.5 | 11.1 | 11.1 | 20.5 |
| sct | 86 | 7 | 72 | 10 | 65 | 6 | 65 | 6 | 6 | 65 | 16.3 | −42.9 | 24.4 | 14.3 | 24.4 | 14.3 | 14.3 | 24.4 |
| cc | 49 | 5 | 44 | 5 | 43 | 5 | 43 | 5 | 5 | 43 | 10.2 | 0.0 | 12.2 | 0.0 | 12.2 | 0.0 | 0.0 | 12.2 |
| cm150a | 54 | 8 | 46 | 8 | 46 | 9 | 37 | 6 | 6 | 37 | 14.8 | 0.0 | 14.8 | −12.5 | 31.5 | 25.0 | 25.0 | 31.5 |
| mux | 55 | 7 | 46 | 7 | 46 | 9 | 37 | 6 | 6 | 37 | 16.4 | 0.0 | 16.6 | −28.6 | 32.7 | 14.3 | 14.3 | 32.7 |
| ttt2 | 187 | 11 | 154 | 11 | 145 | 11 | 144 | 10 | 10 | 144 | 17.6 | 0.0 | 17.6 | 0.0 | 22.5 | 9.0 | 9.0 | 23.0 |
| i1 | 54 | 7 | 41 | 8 | 36 | 6 | 35 | 6 | 6 | 35 | 24.0 | −14.3 | 33.3 | 14.3 | 35.2 | 14.3 | 14.3 | 35.2 |
| lal | 123 | 7 | 95 | 9 | 82 | 8 | 64 | 9 | 8 | 82 | 22.8 | −28.6 | 33.3 | −14.3 | 48.0 | −28.6 | −14.3 | 33.3 |
| pcler8 | 107 | 11 | 90 | 8 | 80 | 9 | 80 | 9 | 8 | 90 | 15.9 | 27.3 | 25.2 | 18.2 | 25.2 | 18.2 | 27.3 | 15.9 |
| frg1 | 196 | 17 | 111 | 23 | 105 | 18 | 102 | 17 | 17 | 102 | 43.4 | −35.3 | 46.4 | −5.9 | 48.0 | 0.0 | 0.0 | 48.0 |
| c8 | 124 | 8 | 115 | 8 | 112 | 7 | 108 | 8 | 7 | 112 | 10.2 | 0.0 | 12.5 | 12.5 | 15.6 | 0.0 | 12.5 | 12.5 |
| term1 | 352 | 12 | 174 | 16 | 106 | 11 | 89 | 10 | 10 | 89 | 50.1 | −33.3 | 69.9 | 8.3 | 74.7 | 16.7 | 16.7 | 74.7 |
| unreg | 84 | 5 | 84 | 4 | 84 | 5 | 84 | 5 | 5 | 84 | 0.0 | 20.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| k2 | 1602 | 18 | 1313 | 19 | 1301 | 19 | 1193 | 19 | 19 | 1193 | 18.0 | −5.6 | 18.8 | −5.6 | 25.5 | −5.6 | −5.6 | 25.5 |
| cht | 121 | 4 | 120 | 4 | 120 | 4 | 120 | 4 | 4 | 120 | 0.8 | 0.0 | 0.8 | 0.0 | 0.8 | 0.0 | 0.0 | 0.8 |
| x1 | 573 | 12 | 320 | 13 | 264 | 11 | 253 | 11 | 11 | 253 | 44.2 | −8.3 | 53.9 | 8.3 | 55.8 | 8.3 | 8.3 | 55.8 |
| example2 | 285 | 9 | 259 | 9 | 247 | 10 | 241 | 9 | 9 | 241 | 9.1 | 0.0 | 13.3 | −11.1 | 15.4 | 0.0 | 0.0 | 15.4 |
| apex6 | 984 | 16 | 701 | 14 | 662 | 17 | 662 | 17 | 15 | 677 | 28.8 | 12.5 | 32.7 | −6.3 | 32.7 | −6.3 | 6.3 | 31.2 |
| frg2 | 759 | 14 | 672 | 15 | 582 | 14 | 568 | 15 | 13 | 600 | 11.5 | −7.1 | 23.3 | 0.0 | 25.2 | −7.1 | 7.1 | 20.9 |
| i2 | 395 | 14 | 209 | 18 | 209 | 13 | 209 | 13 | 13 | 209 | 47.0 | −28.6 | 47.0 | 7.1 | 47.0 | 7.1 | 7.1 | 47.0 |
| Average reduction% | | | | | | | | | | | 22.0% | −7.5% | 31.8% | 5.7% | 36.1% | 6.9% | 11.1% | 34.3% |

as well as 6.9% in level counts, whereas Kong's method and MALS have an average reduction of 31.8% and 22.0% in the number of gates, respectively. When the MLUT method is targeted to optimize the level counts, there is an average reduction of 11.1% in the number of levels as well as 34.3% in the number of gates, whereas Kong's method and MALS have an average reduction of 5.7% and −7.5% in level counts, respectively. It can be noticed that all methods give better average reduction results for gates and levels except the MALS method which results in a worse average reduction for level counts as compared to the AND/OR mapping method. Even though the MLUT method results in the highest average reduction for gate and level counts compared to other methods, it does not result in the optimal majority networks for some circuits. For example, the obtained majority network for the benchmark circuit cm152a using MLUT when targeted to optimize either majority gates or levels, requires six levels, whereas it can be realized with five levels as obtained from Kong's method.

As a result, it can be observed from Table 2 and Table 3 that none of the comprehensive synthesis methods can generate the optimal majority/minority logic networks in terms of all optimization factors for all cases. However, some of these methods can result in best solutions in terms of some optimization factors for three-variable or multi-variable Boolean functions. Table 4 shows the capability of each synthesis method to optimize gates, levels, inverters and gate inputs for all cases of three-variable and multi-variable Boolean functions. From the table, it can be seen that Kong's method and MLUT can generate the optimal majority networks in terms of gates, levels and inverters for all cases of three-feasible networks. However, none of these methods can generate the optimal majority networks in terms of gate inputs for all cases. For Boolean functions with more than three variables, only the MLUT method can synthesize the optimized majority networks in terms of gates and inverters for all cases. Although these results are the best compared to other methods, they are not guaranteed to be optimal. For levels and gate inputs, none of the synthesis methods can give the optimal solutions in terms of these factors for all cases of multi-variable Boolean functions.

Even though these methods result in the best majority networks in terms of some or all optimization factors for all cases, these networks are not guaranteed to be optimal especially while synthesizing multi-output Boolean functions. The process of selecting the optimal majority network for multi-output Boolean functions is not considered in any of the three synthesis methods, which is a very

**Table 4.** Optimization capability analysis for best comprehensive synthesis methods.

| Method | Decomposition | | | | Optimization targets | | | |
|---|---|---|---|---|---|---|---|---|
| | Gates | Levels | Inverters | Gate inputs | Gates | Levels | Inverters | Gate inputs |
| MALS [20] | No | No | No | No | No | No | No | No |
| Kong's [21] | Yes | Yes | Yes | No | No | No | No | No |
| MLUT [26] | Yes | Yes | Yes | No | Yes | No | Yes | No |

serious drawback. For a multi-output Boolean function, by synthesizing the equivalent majority expression for each output separately, which is performed in the three methods, the obtained majority expression can be the optimal in terms of all optimization factors for this output. However, the final majority network realized from these expressions is only optimal in terms of levels, which is the maximum number of levels used in these expressions. For the number of majority gates, inverters, and gate inputs, the final network is not always the optimal solution in terms of these factors. In other words, the number of gates, inverters, and gate inputs used in a majority network obtained from one of these methods for a multi-output Boolean network can be further reduced. To clarify this point, consider a Boolean network $N$ with two outputs, *i.e.*, $F = x_1x_2x_3 + x_1'x_3' + x_1x_2'x_3'$ and $G = x_1x_2x_3 + x_1'x_2'x_3$. For the output $F$, one of its equivalent optimal majority expressions is $F = M\left(M\left(x_1, 0, x_3\right), M\left(x_1, x_2, x_3\right)', M\left(x_1, x_2, x_3'\right)\right)$. For the output $G$, two of its equivalent majority expressions are $G_1 = M\left(M\left(x_1, x_2, x_3\right)', x_3, M\left(x_1, x_2, x_3'\right)\right)$ and $G_2 = M\left(M\left(x_1, x_2, x_3\right)', x_1, M\left(x_1', x_2, x_3\right)\right)$. It can be noticed that both majority expressions for the output $G$ have the same number of gates, levels, inverters, and gate inputs as 3, 2, 2, and 9, respectively. Now, the final majority network for $N$ can be realized by selecting either majority expressions $\left(F, G_1\right)$ or $\left(F, G_2\right)$. However, these networks are different in terms of some optimization factors. For the network $\left(F, G_1\right)$, it has 5 gates, 2 levels, 2 inverters, and 14 gate inputs, whereas the second network $\left(F, G_2\right)$ has 6 gates, 2 levels, 3 inverters, and 16 gate inputs. From the two solutions, it can be seen that the number of levels is the only factor that does not change. However, the second network $\left(F, G_1\right)$ has the minimum number of gates, inverters, and gate inputs. Therefore, the best solution for network $N$ is $\left(F, G_1\right)$. Consequently, it can be seen that this is an important process that can provide further reduction and give better results in terms of different optimization factors.

As discussed earlier, since the different characteristics of nanotechnologies and their logic devices implementations can affect the optimization priorities given to different factors such as gates, levels, inverters, etc., a majority/minority logic network generated from the existing synthesis methods is not guaranteed to be the best solution for all nanotechnologies. Therefore, there is a strong need for developing an efficient majority/minority logic synthesis method that can synthesize the optimal majority/minority networks in terms of all optimization factors for any majority/minority-based nanotechnology.

## 5. Conclusions

Due to the physical limitations of CMOS technology, many emerging nanoscale technologies such as quantum-dot cellular automate (QCA), single electron tunneling (SET), tunneling phase logic (TPL), spintronic devices, etc., have been proposed and considered as possible replacements for CMOS. As known, CMOS

technology uses logic NAND, NOR and NOT gates to implement circuits. However, in post-CMOS nanotechnologies, majority and/or minority gates are the fundamental logic units used to implement Boolean functions. Since traditional reduction methods cannot result in optimal majority or minority logic networks, several papers have introduced different synthesis methods based on different principles. In this paper, we give a comprehensive review of majority/minority logic network synthesis methods that are capable of synthesizing multi-input multi-output Boolean functions. Each of these methods is discussed in detail. We also compare and discuss the obtained results from these methods based on different optimization factors such as the number of gates, the number of levels, etc. From this comparison, we observe that the existing techniques can give sub-optimal solutions. However, none of these methods results in the optimal majority/minority logic networks in terms of all optimization factors for all cases.

For future work in the majority/minority logic synthesis methods, it is suggested that the synthesis method should be developed to synthesis the equivalent majority and minority logic circuits for multi-input multi-output Boolean functions based on optimization techniques that can lead to optimal majority/minority circuits for more than four-feasible networks. In addition, as discussed previously, for a multi-output Boolean function, by generating the optimal majority circuit for each output Boolean function separately, the final majority circuit may not always be optimal. Therefore, it is better to synthesize the majority or minority circuit for any output function with consideration of the other output Boolean functions. Moreover, it suggested that the synthesis method should be developed to generate the optimal majority circuit in terms of all optimization factors based on the given priorities. By developing a method that can synthesize the majority circuits based on different priorities such as gates, levels, inverters, and gate inputs, the method can be used to generate the equivalent circuits for any majority and/or minority-based nanotechnologies.

## Acknowledgements

The authors are thankful to the anonymous reviewers for their comments and suggestions to improve the paper.

## References

[1] 2013 International Technology Roadmap for Semiconductors (ITRS). http://www.semiconductors.org

[2] Lent, C.S., Tougaw, P.D., Porod, W. and Bernstein, G.H. (1993) Quantum Cellular Automata. *Nanotechnology*, **4**, 49. https://doi.org/10.1088/0957-4484/4/1/004

[3] Tougaw, P.D. and Lent, C.S. (1994) Logical Devices Implemented Using Quantum Cellular Automata. *Journal of Applied Physics*, **75**, 1818-1825. https://doi.org/10.1063/1.356375

[4] Lent, C.S. and Tougaw, P.D. (1997) A Device Architecture for Computing with Quantum Dots. *Proceedings of the IEEE*, **85**, 541-557. https://doi.org/10.1109/5.573740

[5] Porod, W. (1997) Quantum-Dot Devices and Quantum-Dot Cellular Automata. *International Journal of Bifurcation and Chaos*, **7**, 2199-2218. https://doi.org/10.1142/S0218127497001606

[6] Snider, G., Orlov, A., Amlani, I., Zuo, X., Bernstein, G., Lent, C., Merz, J. and Porod, W. (1999) Quantum-Dot Cellular Automata: Review and Recent Experiments. *Journal of Applied Physics*, **85**, 4283-4285. https://doi.org/10.1063/1.370344

[7] Walus, K., Jullien, G.A. and Dimitrov, V.S. (2003) Computer Arithmetic Structures for Quantum Cellular Automata. *Conference Record of the* 37*th Asilomar Conference on Signals, Systems and Computers*, Pacific Grove, CA, 9-12 November 2003, 1435-1439. https://doi.org/10.1109/ACSSC.2003.1292223

[8] Oya, T., Asai, T., Fukui, T. and Amemiya, Y. (2002) A Majority-Logic Nanodevice Using a Balanced Pair of Single-Electron Boxes. *Journal of Nanoscience and Nanotechnology*, **2**, 333-342. https://doi.org/10.1166/jnn.2002.108

[9] Oya, T., Asai, T., Fukui, T. and Amemiya, Y. (2003) A Majority-Logic Device Using an Irreversible Single-Electron Box. *IEEE Transactions on Nanotechnology*, **2**, 15-22. https://doi.org/10.1109/TNANO.2003.808507

[10] Fahmy, H.A.H. and Kiehl, R.A. (1999) Complete Logic Family Using Tunneling-Phase-Logic Devices. *Proceedings of the International Conference on Microelectronics*, Kuwait City, November 1999, 153-156.

[11] Nikonov, D.E. and Young, I.A. (2013) Overview of Beyond-CMOS Devices and a Uniform Methodology for Their Benchmarking. *Proceedings of the IEEE*, **101**, 2498-2533. https://doi.org/10.1109/JPROC.2013.2252317

[12] Miller, H.S. and Winder, R.O. (1962) Majority-Logic Synthesis by Geometric Methods. *IRE Transactions on Electronic Computers*, ***EC*-11**, 89-90. https://doi.org/10.1109/TEC.1962.5219329

[13] Akers, S.B. (1962) Synthesis of Combinational Logic Using Three-Input Majority Gates. *Proceedings of the Third Annual Symposium on Switching Circuit Theory and Logical Design*, Chicago, IL, 7-12 October 1962, 149-158. https://doi.org/10.1109/FOCS.1962.16

[14] Muroga, S. (1971) Threshold Logic and Its Applications. Wiley, New York.

[15] Rumi, Z., Walus, K., Wang, W. and Jullien, G.A. (2004) A Method of Majority Logic Reduction for Quantum Cellular Automata. *IEEE Transactions on Nanotechnology*, **3**, 443-450. https://doi.org/10.1109/TNANO.2004.834177

[16] Walus, K., Schulhof, G., Jullien, G.A., Zhang, R. and Wang, W. (2004) Circuit Design Based on Majority Gates for Applications with Quantum-Dot Cellular Automata. *Conference Record of the Thirty-Eighth Asilomar Conference on Signals, Systems and Computers*, Pacific Grove, CA, 7-10 November 2004, 1354-1357.

[17] Bonyadi, M.R., Azghadi, S.M.R., Rad, N.M., Navi, K. and Afjei, E. (2007) Logic Optimization for Majority Gate-Based Nanoelectronic Circuits Based on Genetic Algorithm. *International Conference on Electrical Engineering*, *ICEE*'07, Lahore, 11-12 April 2007, 1-5.

[18] Rai, S. (2008) Majority Gate Based Design for Combinational Quantum Cellular Automata (QCA) Circuits. 40*th Southeastern Symposium on System Theory*, New Orleans, LA, 16-18 March 2008, 222-224. https://doi.org/10.1109/SSST.2008.4480225

[19] Huo, Z., Zhang, Q., Haruehanroengra, S. and Wang, W. (2006) Logic Optimization for Majority Gate-Based Nanoelectronic Circuits. 2006 *IEEE International Symposium on Circuits and Systems*, Island of Kos, 21-24 May 2006, 1307-1310.

[20] Zhang, R., Gupta, P. and Jha, N.K. (2007) Majority and Minority Network Synthesis

with Application to QCA-, SET-, and TPL-Based Nanotechnologies. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, **26**, 1233-1245. https://doi.org/10.1109/TCAD.2006.888267

[21] Kong, K., Shang, Y. and Lu, R. (2010) An Optimized Majority Logic Synthesis Methodology for Quantum-Dot Cellular Automata. *IEEE Transactions on Nanotechnology*, **9**, 170-183. https://doi.org/10.1109/TNANO.2009.2028609

[22] Wang, P., Niamat, M. and Vemuru, S. (2013) Minimal Majority Gate Mapping of Four-Variable Functions for Quantum-Dot Cellular Automata. In: Iniewski, K., Ed., *Nanoelectronic Device Applications Handbook*, CRC Press, Boca Raton, Florida, 263-280.

[23] Mitchell, M. (1998) An Introduction to Genetic Algorithms. MIT Press, Cambridge, Massachusetts.

[24] Holland, J.H. (1992) Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence. U Michigan Press, Ann Arbor, Michigan.

[25] Sentovich, E.M., Singh, K.J., Lavagno, L., Moon, C., Murgai, R., Saldanha, A., Savoj, H., Stephan, P.R., Brayton, R.K. and Sangiovanni-Vincentelli, A. (1992) SIS: A System for Sequential Circuit Synthesis. Technical Report No. UCB/ERL M92/41. University of California, Berkeley.

[26] Wang, P., Niamat, M.Y., Vemuru, S.R., Alam, M. and Killian, T. (2015) Synthesis of Majority/Minority Logic Networks. *IEEE Transactions on Nanotechnology*, **14**, 473-483. https://doi.org/10.1109/TNANO.2015.2408330

[27] Orlov, A.O., Amlani, I., Toth, G., Lent, C.S., Bernstein, G.H. and Snider, G.L. (1999) Experimental Demonstration of a Binary Wire for Quantum-Dot Cellular Automata. *Applied Physics Letters*, **74**, 2875-2877. https://doi.org/10.1063/1.124043

[28] Orlov, A.O., Amlani, I., Kummamuru, R.K., Ramasubramaniam, R., Toth, G., Lent, C.S., Bernstein, G.H. and Snider, G.L. (2000) Experimental Demonstration of Clocked Single-Electron Switching in Quantum-Dot Cellular Automata. *Applied Physics Letters*, **77**, 295-297. https://doi.org/10.1063/1.126955

[29] Nikonov, D.E., Bourianoff, G.I. and Ghani, T. (2011) Proposal of a Spin Torque Majority Gate Logic. *IEEE Electron Device Letters*, **32**, 1128-1130. https://doi.org/10.1109/LED.2011.2156379

[30] Bourianoff, G.I. and Nikonov, D.E. (2011) Recent Progress, Opportunities and Challenges for Beyond CMOS Information Processing Technologies. *ECS Transactions*, **35**, 43-53. https://doi.org/10.1149/1.3568847

[31] Behin-Aein, B., Datta, D., Salahuddin, S. and Datta, S. (2010) Proposal for an All-Spin Logic Device with Built-In Memory. *Nature Nanotechnology*, **5**, 266-270. https://doi.org/10.1038/nnano.2010.31

[32] Behin-Aein, B., Sarkar, A., Srinivasan, S. and Datta, S. (2011) Switching Energy-Delay of All Spin Logic Devices. *Applied Physics Letters*, **98**, Article ID: 123510. https://doi.org/10.1063/1.3567772

[33] Mankalale, M.G. and Sapatnekar, S.S. (2016) Optimized Standard Cells for All-Spin Logic. *ACM Journal on Emerging Technologies in Computing Systems* (*JETC*), **13**, 21. https://doi.org/10.1145/2967612

[34] Krivorotov, I. and Markovic, D. (2011) STT Oscillators/Memory. MIND Annual Review and NRI Benchmarking Workshop, Notre Dame, IN, USA, 16-18.

[35] Khitun, A. and Wang, K.L. (2005) Nano Scale Computational Architectures with Spin Wave Bus. *Superlattices and Microstructures*, **38**, 184-200.

https://doi.org/10.1016/j.spmi.2005.07.001

[36] Bernstein, K., Cavin, R.K., Porod, W., Seabaugh, A. and Welser, J. (2005) Device and Architecture Outlook for beyond CMOS Switches. *Proceedings of the IEEE*, **98**, 2169-2184. https://doi.org/10.1109/JPROC.2010.2066530

[37] Klingler, S., Pirro, P., Bracher, T., Leven, B., Hillebrands, B. and Chumak, A.V. (2014) Design of a Spin-Wave Majority Gate Employing Mode Selection. *Applied Physics Letters*, **105**, Article ID: 152410. https://doi.org/10.1063/1.4898042

[38] Zografos, O., Raghavan, P., Amaru, L., Soree,´ B., Lauwereins, R., Radu, I., Verkest, D. and Thean, A. (2014) System-Level Assessment and Area Evaluation of Spin Wave Logic Circuits. 2014 *IEEE/ACM International Symposium on Nanoscale Architectures* (*NANOARCH*), Paris, 8-10 July 2014, 25-30. https://doi.org/10.1109/NANOARCH.2014.6880475

[39] Khitun, A. and Wang, K.L. (2011) Non-Volatile Magnonic Logic Circuits Engineering. *Journal of Applied Physics*, **110**, Article ID: 034306. https://doi.org/10.1063/1.3609062

[40] Kostylev, M., Serga, A., Schneider, T., Leven, B. and Hillebrands, B. (2005) Spin-Wave Logical Gates. *Applied Physics Letters*, **110**, Article ID: 153501. https://doi.org/10.1063/1.2089147

[41] Amaru, L., Gaillardon, P.-E., Mitra, S. and De Micheli, G. (2015) New Logic Synthesis as Nanotechnology Enabler. *Proceedings of the IEEE*, **103**, 2168-2195. https://doi.org/10.1109/JPROC.2015.2460377

[42] Cowburn, R.P. and Welland, M.E. (2000) Room Temperature Magnetic Quantum Cellular Automata. *Science*, **287**, 1466-1468. https://doi.org/10.1126/science.287.5457.1466

[43] Zhu, J., Zhang, L., Dong, S. and Wang, E. (2013) Four-Way Junction-Driven DNA Strand Displacement and Its Application in Building Majority Logic Circuit. *ACS Nano*, **7**, 10211-10217. https://doi.org/10.1021/nn4044854

[44] Li, W., Yang, Y., Yan, H. and Liu, Y. (2013) Three-Input Majority Logic Gate and Multiple Input Logic Circuit Based on DNA Strand Displacement. *Nano Letters*, **13**, 2980-2988. https://doi.org/10.1021/nl4016107

[45] George, A.K. and Singh, H. (2016) Three-Input Majority Gate Using Spatially Localised DNA Hairpins. *IEEE Transactions on NanoBioscience*, **15**, 928-938. https://doi.org/10.1109/TNB.2016.2623218

[46] Schwierz, F. (2010) Graphene Transistors. *Nature Nanotechnology*, **5**, 487-496. https://doi.org/10.1038/nnano.2010.89

[47] Yang, H., Heo, J., Park, S., Song, H.J., Seo, D.H., Byun, K.-E., Kim, P., Yoo, I., Chung, H.-J. and Kim, K. (2010) Graphene Barristor, a Triode Device with a Gate-Controlled Schottky Barrier. *Science*, **336**, 1140-1143. https://doi.org/10.1126/science.1220527

[48] Miryala, S., Tenace, V., Calimera, A., Macii, E., Poncino, M., Amaru, L., De Micheli, G. and Gaillardon, P.-E. (2015) Exploiting the Expressive Power of Graphene Reconfigurable Gates via Post-Synthesis Optimization. 25*th Great Lakes Symposium on VLSI* (*GLSVLSI*), Pittsburgh, Pennsylvania, 20-22 May 2015, 39-44. https://doi.org/10.1145/2742060.2742098

[49] Linn, E., Rosezin, R., Kugeler, C. and Waser, R. (2010) Complementary Resistive Switches for Passive Nanocrossbar Memories. *Nature Materials*, **9**, 403-406. https://doi.org/10.1038/nmat2748

[50] Fackenthal, R., Kitagawa, M., Otsuka, W., Prall, K., Mills, D., Tsutsui, K., Javanifard,

J., Tedrow, K., Tsushima, T., Shibahara, Y., *et al.* (2014) 19.7 A 16Gb ReRAM with 200MB/s Write and 1GB/s Read in 27 nm Technology. 2014 *IEEE International Solid-State Circuits Conference Digest of Technical Papers* (*ISSCC*), San Francisco, CA, 9-13 February 2014, 338-339. https://doi.org/10.1109/ISSCC.2014.6757460

[51] Sheu, S.-S., Chang, M.-F., Lin, K.-F., Wu, C.-W., Chen, Y.-S., Chiu, P.-F., Kuo, C.-C., Yang, Y.-S., Chiang, P.-C., Lin, W.-P., *et al.* (2011) A 4Mb Embedded SLC Resistive-RAM Macro with 7.2 ns Read-Write Random-Access Time and 160 ns MLC-Access Capability. IEEE Solid-State Circuits Conference Digest of Technical Papers, February 2011, 200-202.

[52] Lin, Y.-M., Appenzeller, J., Knoch, J. and Avouris, P. (2005) High-Performance Carbon Nanotube Field Effect Transistor with Tunable Polarities. *IEEE Transactions on Nanotechnology*, **4**, 481-489. https://doi.org/10.1109/TNANO.2005.851427

[53] Appenzeller, J. (2008) Carbon Nanotubes for High-Performance Electronics Progress and Prospect. *Proceedings of the IEEE*, **96**, 201-211. https://doi.org/10.1109/JPROC.2007.911051

[54] Navi, K., Momeni, A., Sharifi, F. and Keshavarzian, P. (2009) Two Novel Ultra High Speed Carbon Nanotube Full-Adder Cells. *IEICE Electronics Express*, **6**, 1395-1401. https://doi.org/10.1587/elex.6.1395

[55] Navi, K., Sharifi, F., Momeni, A. and Keshavarzian, P. (2009) Ultra High Speed CNFET Full-Adder Cell Based on Majority Gates. *IEICE Transactions on Electronics*, **93**, 932-934.

[56] Dadda, L. (1963) Information Processing: Proceedings of the IFIP Congress. North Holland, The Netherlands.

[57] Lisanke, R. (1988) Logic Synthesis and Optimization Benchmarks User Guide Version 2.0. Tech Report, Microelectronics Center North Carolina, Research Triangle Park.