Majority Logic Circuit Minimization Using Node Addition and Removal

Chang-Cheng Ko, Chia-Chun Lin, Yung-Chih Chen, *Member, IEEE*, and Chun-Yao Wang, *Member, IEEE*

***Abstract*—Quantum-dot cellular automata (QCA) is considered as a promising emerging technology due to its low power dissipation and high device density. Since the majority function is the main operation in QCA circuits, minimizing the number of majority gates in QCA circuits is crucial to the corresponding QCA circuit minimization. A previous work used the node-merging technique to replace one target node with an existing substitute node in majority circuits for optimization. However, this technique may fail when no substitute nodes exist for a target node. In this article, we propose an enhanced optimization technique for majority circuits by adding a new node into the circuits and removing the target node and its fanin nodes. The experimental results show that this technique improves the results of the node-merging technique on a set of EPFL logic synthesis benchmarks. Additionally, this enhanced technique can work together with other optimization techniques. The circuit size reduction in the integrated approach reaches 1.26 times as compared to the results using the node-merging technique.**

***Index Terms*—Logic implication, logic optimization, majority logic, node addition and removal (NAR), node-merging, observability do not care (ODC).**

I. Introduction

With the advances of nanotechnologies, the idea of using different alternatives to CMOS transistors as the building elements of very large scale integrated designs (VLSI) has been proposed [26]. Among these nanotechnologies, quantum-dot cellular automata (QCA) is considered as a promising one due to its low power dissipation and high device density [28], [29], [33]. The operating mechanism of QCA offers low power and high-speed computation since there is no interconnection in signal paths of cells. That is, the Coulombic interactions among cells are beneficial to power dissipation and state transitions [33], [40].

The fundamental elements for building a QCA circuit are quantum-dot cells. A quantum-dot cell is composed of four quantum dots and two free electrons. By Coulombic repulsion, these electrons will occupy at diagonal positions with respect to each other in one cell. Fig. 1 shows two different configurations of electrons in a quantum-dot cell, which present two distinguishable states. These two configurations are resulted from the polarization of a quantum-dot cell, denoted as P. Binary logic information can be encoded by using this cell polarization, i.e., $P = +1$ for "1" and $P = -1$ for "0."

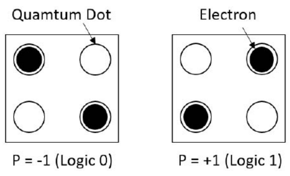

Fig. 1. QCA cells and its binary information.

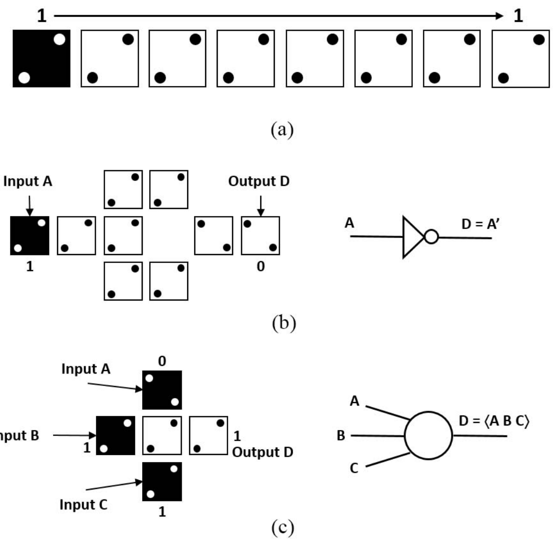

Fig. 2. (a) QCA wire. (b) QCA inverter. (c) QCA majority gate.

Quantum-dot cells are used for constructing basic computation units of QCA, such as wires, inverters, and majority gates. In Fig. 2(a), a QCA wire comprises a line of QCA cells for signal propagation from the input cell to the output cell [19], [26], [41]. An example of the QCA inverter is shown in Fig. 2(b). By placing electrons in a diagonal position, the signal starts from A with the value of "1" and reaches at D with the value of "0". Note that QCA inverters can also be realized by only two QCA cells, the layout of the inverter we showed here is a more general and robust design and can provide better output polarization [5], [23]. An example of the

Manuscript received March 19, 2020; revised July 28, 2020 and October 20, 2020; accepted February 10, 2021. Date of publication February 18, 2021; date of current version February 21, 2022. This work was supported by the Ministry of Science and Technology of Taiwan under Grant MOST 108-2218-E-007-061, Grant MOST 109-2221-E-007-082-MY2, Grant MOST 109-2221-E-155-047-MY2, and Grant MOST 109-2224-E-007-005. This article was recommended by Associate Editor Z. Zhang. *(Corresponding author: Chia-Chun Lin.)*

Chang-Cheng Ko, Chia-Chun Lin, and Chun-Yao Wang are with the Department of Computer Science, National Tsing Hua University, Hsinchu 30013, Taiwan (e-mail: s107062550@m107.nthu.edu.tw; chiachun-lin@gapp.nthu.edu.tw; wcyao@cs.nthu.edu.tw).

Yung-Chih Chen is with the Department of Computer Science and Engineering, Yuan Ze University, Taoyuan 32003, Taiwan (e-mail: ycchen.cse@saturn.yzu.edu.tw).

Digital Object Identifier 10.1109/TCAD.2021.3060648

QCA majority gate is shown in Fig. 2(c), which comprises three input cells and one output cell. The value of the output signal will obey the majority rule depending on the values of input signals. For example, if two of three inputs are "1," like the input cells B and C, the majority gate will output the value of "1."

Since the majority function is the main operation in QCA circuits, minimizing the number of majority gates in QCA circuits is crucial to the corresponding QCA circuit minimization. Recently, many studies about majority logic have been proposed [1]–[3], [15]–[17], [21], [26], [36], [37], [39], [44]. Kong *et al.* [26] and Zhang *et al.* [44] proposed methods to convert a given Boolean function into its corresponding minimum cost majority expression. Amarú *et al.* [1], [2] proposed a new Boolean logic representation—majority-inverter-graph (MIG), and its axiomatic system for optimizing delay and power of logic circuits. Furthermore, the authors further proposed an MIG-based optimization method by exploiting the error masking property of majority functions [3]. After that, Soeken *et al.* [36] proposed an MIG optimization algorithm based on functional hashing. Moreover, one node-merging approach for MIGs using Boolean SAT has been presented under the name—functionally-reduced MIGs (FRMIGs) [20], [37] for lookup tables (LUTs) network. Besides, a majority-based logic synthesis technique with a restricted fanout number complying with the technology constraints of QCA was also proposed to synthesize the network [39]. Recently, Neutzling *et al.* [31] proposed a novel synthesis flow—maj-*n* —for majority circuits, which was able to handle gates with an arbitrary number of inputs. Riener *et al.* [34], [35] proposed a Boolean resubstitution approach for MIGs, which re-expresses the logic function of a node using existing nodes in the network to optimize the size of logic circuits.

These years, a compact logic representation for a more efficient logic synthesis—XOR-majority-graph (XMG) has been introduced [21]. Then, Chu *et al.* [15] proposed a decomposition algorithm exploiting MAJ to reduce the area or depth of XMGs. Furthermore, the authors proposed an algebraic rewriting on XMGs by combining the axiomatic system [1] and XOR primitives for gaining more optimization opportunities [16]. Despite lots of studies on majority logic have been proposed, only few of them aim at the minimization of the size of majority circuits. A work exploiting the node-merging technique [10], [11] considering observability don't cares (ODCs) for optimizing majority circuits has been proposed such that the corresponding QCA circuits are minimized [17].

The node-merging technique is a logic restructuring technique that aims at replacing a target node n_t with a substitute node n_s to minimize logic circuits without changing the overall functionalities. Chen and Wang [12], [13] proposed an enhanced algorithm called node addition and removal (NAR), which adds a new node into the circuit to replace a target node. NAR belongs to the category of redundancy addition and removal (RAR), which adds redundancies to the original circuits for removing the target objects, such as wires or nodes. This technique can be used in the applications of area reduction, reliability improvement, or timing optimization of VLSI circuits [6]–[9], [14], [18], [30]. For the application of area reduction, when more than one node can be removed due to a newly added node, the size of the whole circuit will be reduced. However, the original NAR algorithm is only suitable for and-inverter-graphs (AIGs) [22], and cannot be applied to MIGs or XMGs. Thus, in this work, we propose a new NAR algorithm for majority circuits. Its theoretical details are also investigated.

We conducted several experiments on a set of majority logic benchmarks provided by EPFL Integrated Systems Laboratory [50], [51]. The experimental results show that our new NAR approach can further reduce the gate count of the well-optimized benchmarks. We also integrate this work with the state-of-the-art majority logic synthesis tool *CirKit* [49] and the EPFL logic synthesis libraries [38]. The results show that the overall circuit size can be reduced by 15.06% on average, which is 1.26 times compared to the approach that only uses the node-merging technique [17], for the same set of benchmarks. As compared to the approach that only uses the node-merging technique [17], our method also can reduce the circuit size either with or without *CirKit* and the EPFL logic synthesis library.

The main contributions of this work are twofold.

1) We propose the first NAR algorithm for the majority logic circuits.
2) The proposed algorithm can be well integrated with the state-of-the-art majority logic synthesis tool.

The remainder of this article is organized as follows. Section II introduces the preliminaries of this work. Section III presents the proposed NAR algorithm for majority logic circuits. Section IV proposes a technique for accelerating the computation efficiency of NAR, and introduces the algorithm of circuit size reduction for majority circuits. Finally, the experimental results and conclusions are presented in Sections V and VI.

II. Preliminaries

In this section, we review some backgrounds of this work.

A. Background

The *input-controlling* value on an input determines the output of the gate independent of the other inputs. The *input-noncontrolling* value is opposite to the input-controlling value. For example, the input-controlling value of an AND gate is 0 and the input-noncontrolling value of an AND gate is 1.

A *majority function* is an odd-input function that has the output value of v if and only if more than half of the inputs are the value of v. A majority-of-three (MAJ) function consists of three Boolean variables x, y, and z, and is denoted as $\langle xyz\rangle$ [42]. An MAJ can be expressed in disjunctive or conjunctive normal forms as

$$\langle xyz\rangle = xy \vee xz \vee yz = (x \vee y)(x \vee z)(y \vee z). \quad (1)$$

Setting any variable to "1" or "0" in an MAJ will make the other two variables disjunctive or conjunctive, respectively. For example, when $x = 1$ or $x = 0$, $\langle xyz\rangle$ becomes an OR gate as (2) or becomes an AND gate as (3), respectively

$$\langle 1yz \rangle = y \vee z \tag{2}$$
$$\langle 0yz \rangle = yz. \tag{3}$$

An MIG is a directed, acyclic graph consisting of MAJs and inverters as primitives. An XMG is an extension of MIG, which additionally introduces XOR primitives into the circuits. The edges in the MIGs or XMGs can be uncomplemented or complemented, where complemented edges have a dot on them. A complemented edge represents the negation of a node's function.

The *dominators* [24] of a gate g is a set of gates that all the paths from g to any primary outputs (POs) have to pass through. The *side-inputs* of a path from gate g_x to g_y are the inputs of the nodes that are not in the transitive fanout cone (TFO) of g_x. An MAJ gate has two side-inputs in the fault-effect propagation path. To propagate a fault-effect to the POs, these two side-inputs have to be assigned to different values. Thus, these two side-inputs of an MAJ gate form a *side-input pair*, and the different values of these two side-inputs are named as a *noncontrolling pair* [17]. For example, in Fig. 2(c), if we want to observe the fault-effect from the input A at the output D, the side-input pair (B, C) has to be assigned different values, e.g., (B, C) = (0, 1) or (1, 0), to propagate the fault-effect.

A *stuck-at fault* in VLSI testing is a fault model used to represent a manufacturing defect in logic circuits. The model assumes that a faulty wire or faulty gate is stuck at a fixed logic value "0" or "1," referred to *stuck-at 0* (sa0) or *stuck-at 1* (sa1) fault, respectively. A stuck-at fault test is a process to find a test pattern that can distinguish a faulty circuit from a fault-free one. A test pattern needs to *activate* and *propagate* the fault-effect to any POs. A fault is an *untestable* fault if there exists no test pattern that can activate and propagate the fault-effect to any POs in the circuit. An untestable stuck-at fault on a wire or gate is a *redundancy* to the circuit because the fault-effect cannot be observed at any POs. Therefore, an untestable stuck-at v fault can be replaced with a constant value v for circuit optimization.

The *mandatory assignments* (MAs) are the necessary values assigned to some nodes for detecting a fault in the circuit. In Boolean circuits, the MAs are obtained by assigning the inverse value of the faulty value on a target node to activate the fault-effect, and by assigning the noncontrolling values on the side-inputs to propagate the fault-effect. By forward and backward logic implications, more MAs could be further derived. If the MAs of a fault are inconsistent, e.g., $x = 0$ and $x = 1$ are both MAs during the MA derivation process, it means that no test pattern can detect this fault; therefore, this fault is untestable.

B. MA Computation for Majority Circuits

The number of MAs is crucial to the minimization of a logic network in this work. The more MAs we can obtain during the MA computation, the more possibilities we can identify the substitute node for a given target node. However, computing all the MAs for detecting a stuck-at fault in the circuit is an NP-complete problem [27]. Thus, we adopt the dominator-based MA computation method for a trade-off between the quality and efficiency. Moreover, we also integrate the MA computation with the recursive learning technique [27] with the depth of one for more optimization. Recently, Chung *et al.* [17] proposed an MA computation scheme for majority circuit. In this work, we adopt their procedure for MA computation.

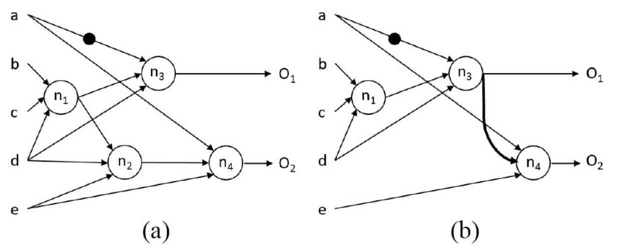

Fig. 3. Example for presenting the node-merging approach. (a) Original circuit. (b) Resultant circuit after replacing n_2 with n_3.

We take the example in Fig. 3 to demonstrate the concepts of stuck-at fault and MA computation. In Fig. 3(a), a, b, c, d, and e are primary inputs (PIs), O_1 and O_2 are POs, and n_1–n_4 are MAJ gates. Given a target node n_2, we want to find the MAs of stuck-at-0 fault on it. As mentioned before, a test pattern needs to activate and propagate the fault-effect to any POs. Thus, the value of n_2 will be assigned to 1 (the inverse value of the faulty value "0") to activate the fault-effect of stuck-at-0 fault. Then, the fault-effect has to be propagated to any POs by setting noncontrolling pairs to all the side-input pairs of all the dominators. We assign (a, e) to (0, 1) and (1, 0) to the side-input pair of dominator n_4 to obtain the value assignments as $\{n_2 = 1, n_4 = 1, a = 0, e = 1, n_3 = 1\}$ and $\{n_2 = 1, n_4 = 1, a = 1, e = 0, n_1 = 1, d = 1, n_3 = 1\}$, respectively. The intersection of these two value assignments are $\{n_2 = 1, n_3 = 1, n_4 = 1\}$. Since the target node n_2 only has one dominator n_4, this set of value assignments are the MAs for detecting the stuck-at-0 fault on n_2.

C. Node-Merging Approach

The node-merging approach was proposed for optimizing logic circuits considering ODCs [10], [11]. It modeled the merge of two nodes as a misplaced-wire error in the circuit. When the error is undetectable, merging these two nodes is safe since it will not affect the overall functionality of the circuit. A sufficient Condition was also proposed for searching legal node mergers as stated in Condition 1 [10], [11].

Condition 1 [10], [11]: Let f denote an error of replacing n_t with n_s. If $n_s = 1$ and $n_s = 0$ are MAs for stuck-at 0 and stuck-at 1 fault tests on n_t, respectively, f is undetectable.

Other popular node-merging techniques include SAT sweeping [25], [32], [46], structural hashing [20], and Boolean resubstitution [34], [35]. SAT sweeping is a method that merges two functionally compatible nodes by simulating a set of random patterns and running SAT solvers [25]. Moreover, the local and global ODC-based algorithms extended the SAT sweeping to increase the number of nodes merged [32], [46]. A similar concept of SAT sweeping technique has been applied for optimizing MIGs [20]. Structural hashing aims at merging two nodes that share the same fanin nodes in the logic network, i.e., structural hashing merges nodes that are structurally equivalent. The structural hashing technique has also been applied for optimizing MIGs [20]. Boolean resubstitution aims at re-expressing the logic function of a node by using other nodes that are already present in the network to optimize the size of the logic circuit. Note that the difference between the Boolean resubstitution approach [34], [35] and the node-merging approach [10], [11] is that Boolean resubstitution often needs many simulations to decide a better candidate node for substitution to reduce the number of nodes in the logic network. Additionally, the Boolean resubstitution approach [34], [35] is limited to a window-based resubstitution and usually does not consider ODCs. The node-merging approach [10], [11] is an ODC-based method considering ODCs in the process of identifying the node mergers with a sufficient condition—Condition 1.

A new node-merging approach was proposed for majority logic circuits by using Condition 1 and the idea of side-input pair [17]. Here, we use the example in Fig. 3 again to demonstrate the node-merging approach for majority circuits. For the nodes n_2 and n_3, replacing n_2 with n_3 may cause an error due to their different functionalities. Because d and n_1 are the common inputs to n_2 and n_3, n_2 and n_3's functions only differ when a and e are the same value v. However, $a = e = v$ also leads the value of node n_4 to v. As a result, the value of n_2 cannot affect n_4, which prevents the different value of n_2 with respect to n_3 from being observed at the POs. Thus, n_2 can be replaced with n_3 without changing the overall functionality. The resultant circuit after the merging is shown in Fig. 3(b), where n_2 is removed and n_3 is used to drive n_4 for n_2.

D. NAR Approach

We also use another example to demonstrate the effectiveness of NAR when the node-merging approach fails. Fig. 4(a) is an MIG, where a, b, c, d, and e are PIs, O_1–O_4 are POs, and n_1–n_8 are MAJ gates. Note that the nodes n_1, n_3–n_5, n_7, and n_8 are equivalent to AND gates when setting one of their fanin signals to a constant value of "0." Given a target node n_6, the node-merging approach fails since it cannot find any substitute nodes in this circuit. Nevertheless, we can add a new node n_9 to replace the node n_6 as shown in Fig. 4(b). After adding n_9 into the circuit, the functionality of this circuit does not change because n_9 does not drive any nodes. However, n_9 meets the requirements of Condition 1 as being a substitute node for n_6. Thus, replacing n_6 with n_9 is safe in terms of functionality. Furthermore, when n_6 is removed, n_1 can be removed as well. The resultant circuit after these addition and removal operations is shown in Fig. 4(c). This example shows that a target node having no substitute node still would be replaced by a newly added node. Meanwhile, the whole circuit can be reduced if this target node has at least a fanin node driving only one node. That is, the removal of the target node also eliminates the target node's fanin node. Therefore, at least two nodes will be removed after adding one node.

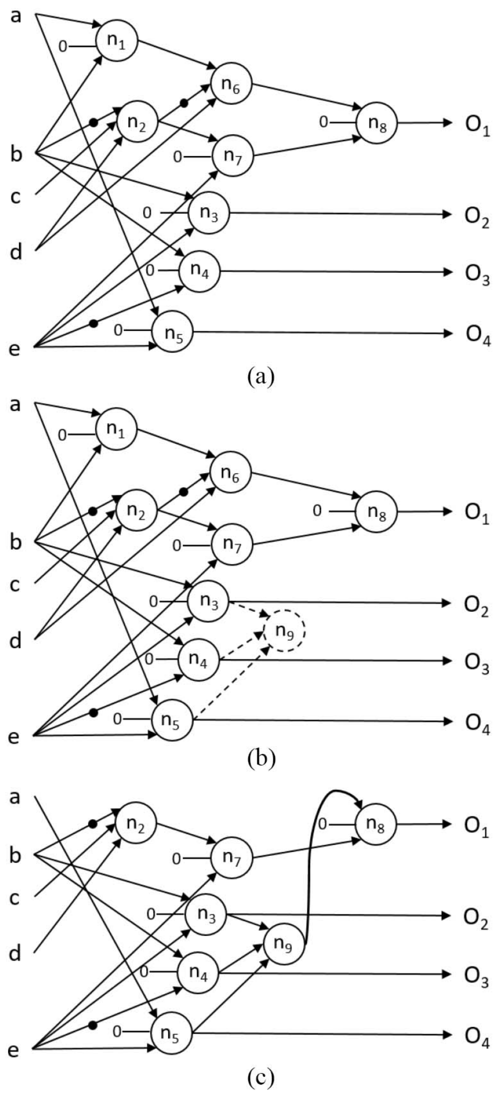

Fig. 4. Example for presenting the NAR approach. (a) Original circuit. (b) Resultant circuit after adding n_9. (c) Resultant circuit after replacing n_6 with n_9, and removing n_1.

III. Proposed Node Addition and Removal Approach for Majority Circuits

In this section, we first propose two sufficient conditions for replacing a target node with an added substitute node. These two conditions can be changed with respect to 16 different types of added substitute nodes. Then, we present the overall algorithm of the proposed NAR approach for majority circuits.

A. Sufficient Conditions in NAR for Majority Circuits

The NAR approach aims at adding an added substitute node for replacing an existing target node in the circuits. Based on Condition 1, we can check whether an added node is able to replace the target node or not. However, it is quite time-consuming if we first add new nodes at all possible locations in the circuit and then use Condition 1 to exhaustively search the substitute nodes for the target node from these new nodes. Therefore, we alternatively focus on finding three existing fanin nodes of an added substitute node satisfying Condition 1.

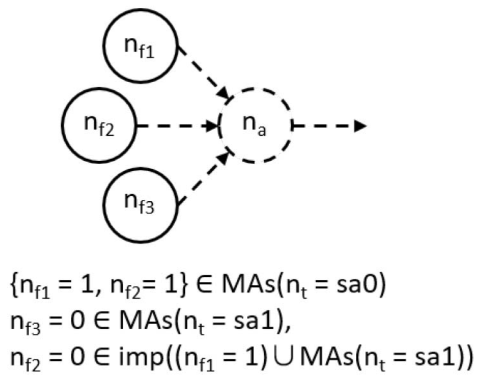
n_{f1}
n_{f2}
n_a
n_{f3}
$\{n_{f1} = 1, n_{f2} = 1\} \in \text{MAs}(n_t = sa0)$
$n_{f3} = 0 \in \text{MAs}(n_t = sa1)$,
$n_{f2} = 0 \in \text{imp}((n_{f1} = 1) \cup \text{MAs}(n_t = sa1))$

Fig. 5. Type 1 added substitute node and its corresponding sufficient conditions.

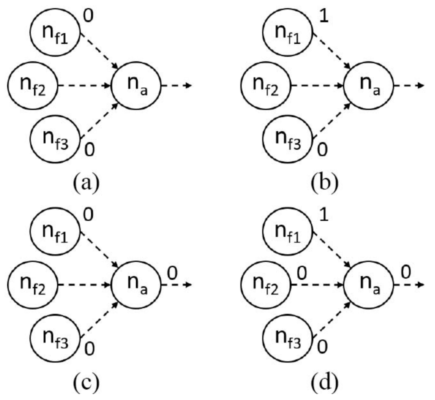

Fig. 6. Scenarios for Condition 3. (a) $T_{n_{f1}=0}$. (b) $T_{n_{f1}=1}$. (c) $\{n_{f3} = 0, n_{f1} = 0\}$ implies $n_a = 0$. (d) $\{n_{f3} = 0, n_{f1} = 1, n_{f2} = 0\}$ implies $n_a = 0$.

For ease of the discussion, we denote the target node as n_t, the added node as n_a, and the three fanin nodes of n_a as n_{f1}, n_{f2}, and n_{f3}. In Fig. 5, the functionality of n_a can be expressed as $\langle n_{f1}n_{f2}n_{f3}\rangle$. Afterward, we will derive two sufficient conditions for n_a—Conditions 2 and 3. If n_a satisfies these two sufficient conditions, n_a will satisfy Condition 1 such that it can replace the target node. In the following paragraphs, we will state Conditions 2 and 3, and explain why they are useful for finding an added substitute node.

Condition 2: Consider a stuck-at 0 fault test on n_t, if $n_{f1} = 1$ and $n_{f2} = 1$ are both MAs for the same fault test, $n_a = 1$ is also an MA for the same fault test.

As we mentioned in Section II, a majority function will output the value v if and only if more than half of its inputs are v. Therefore, if $n_{f1} = 1$ and $n_{f2} = 1$ are both MAs for the stuck-at 0 fault test on n_t, $n_a = 1$ is an MA for the same fault test.

However, when Condition 2 is held for an n_a, n_a only satisfies the first half of Condition 1, i.e., $n_a = 1$ is an MA for the stuck-at 0 fault test on n_t. For n_a to be an added substitute node of n_t, $n_a = 0$ needs to be an MA for the stuck-at 1 fault test on n_t as well. Thus, we further propose another sufficient condition—Condition 3—to make n_a satisfy the second half of Condition 1.

For the ease of discussion, we denote a set of MAs for a stuck-at v fault test on n_t as $\text{MAs}(n_t = sav)$, where v is the value of "0" or "1." We also denote a set of value assignments logically implied by a set of value assignments A as $\text{imp}(A)$.

Condition 3: Consider a stuck-at 1 fault test on n_t, if $n_{f3} = 0$ is a value assignment in $\text{MAs}(n_t = sa1)$, and $n_{f2} = 0$ is a value assignment in $\text{imp}((n_{f1} = 1) \cup \text{MAs}(n_t = sa1))$, $n_a = 0$ is an MA for the same fault test.

Condition 3 is used to ensure $n_a = 0$ is an MA for the stuck-at 1 fault on n_t. When Condition 3 is satisfied, the second half of Condition 1 will be satisfied as well. The statement "$n_a = 0$ is an MA for the stuck-at 1 fault test on n_t" means that all the test patterns detecting this fault make $n_a = 0$. Since n_a is an MAJ, at least two of its inputs are "0" for leading $n_a = 0$. If one fanin node, say n_{f3}, of n_a has been "0" and $\in \text{MAs}(n_t = sa1)$, we need to find another fanin node that is also "0" and is $\in \text{MAs}(n_t = sa1)$ as well.

We denote T as the set of test patterns for detecting the stuck-at 1 fault on n_t, and divide T into two parts as $T_{n_{f1}=0}$ and $T_{n_{f1}=1}$, where $T_{n_{f1}=0}$ and $T_{n_{f1}=1}$ represent the subset of test patterns in T that generate $n_{f1} = 0$, and $n_{f1} = 1$, respectively, as shown in Fig. 6(a) and (b). First, when $n_{f3} = 0$ is a value assignment in $\text{MAs}(n_t = sa1)$, then having another fanin node, say n_{f1}, as "0" will imply $n_a = 0$ as shown in Fig. 6(c). Hence, all the patterns in $\boldsymbol{T_{n_{f1}} = 0}$ will generate $\boldsymbol{n_a} = 0$. Second, we know that $\text{imp}((n_{f1} = 1) \cup \text{MAs}(n_t = sa1))$ is a set of value assignments that are derived by all the test patterns in $T_{n_{f1}=1}$. Since $n_{f3} = 0$ has been a value assignment in $\text{MAs}(n_t = sa1)$, when $n_{f2} = 0$ is also a value assignment in $\text{imp}((n_{f1} = 1) \cup \text{MAs}(n_t = sa1))$, all the patterns in $\boldsymbol{T_{n_{f1}} = 1}$ imply $\boldsymbol{n_a} = 0$ as shown in Fig. 6(d). Therefore, when Condition 3 is held, all the test patterns in $T = T_{n_{f1}=0} \cup T_{n_{f1}=1}$ will generate $n_a = 0$, i.e., $n_a = 0$ is an MA for the stuck-at 1 fault test on n_t.

In summary, when Conditions 2 and 3 are held, $n_a = 1$ and $n_a = 0$ are MAs for the stuck-at 0 and the stuck-at 1 fault test on n_t, respectively, which is the same as Condition 1. Thus, we know that n_a is an added substitute node for n_t. These two sufficient conditions are shown in Fig. 5.

Note that n_{f1}, n_{f2}, and n_{f3} are not any particular fanin nodes of n_a, e.g., n_{f1} is not necessary to be the first fanin node of n_a. When one fanin node has been considered as n_{f1}, the other two fanin nodes will be n_{f2} and n_{f3}. That is, n_{f1}, n_{f2}, and n_{f3} in Condition 3 are exchangeable.

B. Different Types of the Added Substitute Nodes

In Section III-A, we assume that there is no inverter between n_a and its fanin nodes—n_{f1}, n_{f2}, and n_{f3} in the derivation of Conditions 2 and 3. However, we observe that we can modify these two conditions by flipping the values of n_{f1}, n_{f2}, n_{f3}, or the stuck-at fault for different types of the added substitute nodes. These modifications could increase the opportunities for finding more added substitute nodes. We list these different types of added substitute nodes with their corresponding sufficient conditions in Fig. 7.

In Fig. 7, Types 2–8 can be obtained by reversing the values of n_{f1}, n_{f2}, or n_{f3}, respectively. By reversing the value of stuck-at fault in Types 1–8, we can obtain Types 9–16 added substitute nodes.

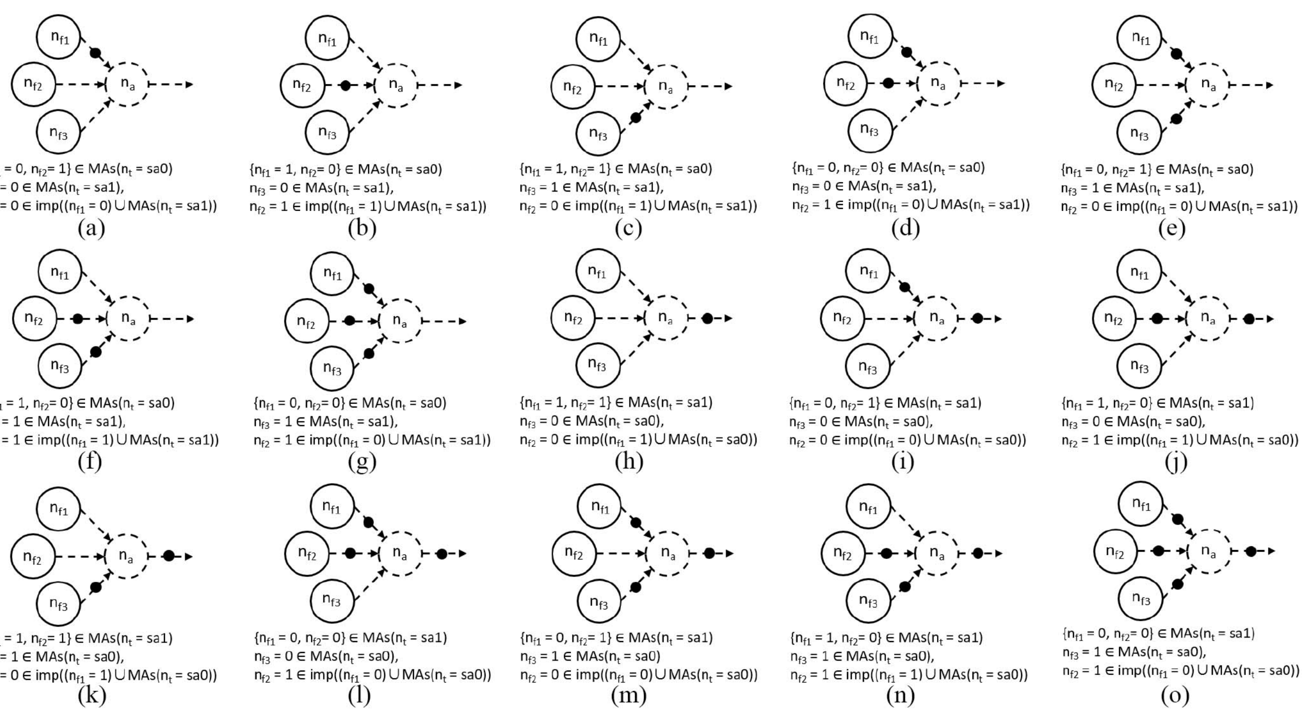
{nf1 = 0, nf2= 1} ∈ MAs(nt = sa0)
nf3 = 0 ∈ MAs(nt = sa1),
nf2 = 0 ∈ imp((nf1 = 0) ∪ MAs(nt = sa1))
(a)
{nf1 = 1, nf2= 0} ∈ MAs(nt = sa0)
nf3 = 0 ∈ MAs(nt = sa1),
nf2 = 1 ∈ imp((nf1 = 1) ∪ MAs(nt = sa1))
(b)
{nf1 = 1, nf2= 1} ∈ MAs(nt = sa0)
nf3 = 1 ∈ MAs(nt = sa1),
nf2 = 0 ∈ imp((nf1 = 1) ∪ MAs(nt = sa1))
(c)
{nf1 = 0, nf2= 0} ∈ MAs(nt = sa0)
nf3 = 0 ∈ MAs(nt = sa1),
nf2 = 1 ∈ imp((nf1 = 0) ∪ MAs(nt = sa1))
(d)
{nf1 = 0, nf2= 1} ∈ MAs(nt = sa0)
nf3 = 1 ∈ MAs(nt = sa1),
nf2 = 0 ∈ imp((nf1 = 0) ∪ MAs(nt = sa1))
(e)
{nf1 = 1, nf2= 0} ∈ MAs(nt = sa0)
nf3 = 1 ∈ MAs(nt = sa1),
nf2 = 1 ∈ imp((nf1 = 1) ∪ MAs(nt = sa1))
(f)
{nf1 = 0, nf2= 0} ∈ MAs(nt = sa0)
nf3 = 1 ∈ MAs(nt = sa1),
nf2 = 1 ∈ imp((nf1 = 0) ∪ MAs(nt = sa1))
(g)
{nf1 = 1, nf2= 1} ∈ MAs(nt = sa1)
nf3 = 0 ∈ MAs(nt = sa0),
nf2 = 0 ∈ imp((nf1 = 1) ∪ MAs(nt = sa0))
(h)
{nf1 = 0, nf2= 1} ∈ MAs(nt = sa1)
nf3 = 0 ∈ MAs(nt = sa0),
nf2 = 0 ∈ imp((nf1 = 0) ∪ MAs(nt = sa0))
(i)
{nf1 = 1, nf2= 0} ∈ MAs(nt = sa1)
nf3 = 0 ∈ MAs(nt = sa0),
nf2 = 1 ∈ imp((nf1 = 1) ∪ MAs(nt = sa0))
(j)
{nf1 = 1, nf2= 1} ∈ MAs(nt = sa1)
nf3 = 1 ∈ MAs(nt = sa0),
nf2 = 0 ∈ imp((nf1 = 1) ∪ MAs(nt = sa0))
(k)
{nf1 = 0, nf2= 0} ∈ MAs(nt = sa1)
nf3 = 0 ∈ MAs(nt = sa0),
nf2 = 1 ∈ imp((nf1 = 0) ∪ MAs(nt = sa0))
(l)
{nf1 = 0, nf2= 1} ∈ MAs(nt = sa1)
nf3 = 1 ∈ MAs(nt = sa0)
nf2 = 0 ∈ imp((nf1 = 0) ∪ MAs(nt = sa0))
(m)
{nf1 = 1, nf2= 0} ∈ MAs(nt = sa1)
nf3 = 1 ∈ MAs(nt = sa0),
nf2 = 1 ∈ imp((nf1 = 1) ∪ MAs(nt = sa0))
(n)
{nf1 = 0, nf2= 0} ∈ MAs(nt = sa1)
nf3 = 1 ∈ MAs(nt = sa0),
nf2 = 1 ∈ imp((nf1 = 0) ∪ MAs(nt = sa0))
(o)
nf1
nf2
nf3
na

Fig. 7. Different types of added substitute nodes and their corresponding sufficient conditions. (a)–(o): Type 2–16.

C. NAR Algorithm for Majority Circuits

The NAR algorithm can be summarized as follows. Given a target node n_t in a majority circuit C. First, we choose an MA in MAs($n_t = sa0$) as n_{f1}. Then the algorithm will search n_{f2} and n_{f3} according to Conditions 2 and 3 based on n_{f1}. Finally, the node driven by n_{f1}, n_{f2}, and n_{f3} is the added substitute node n_a for n_t. The NAR algorithm is shown in Algorithm 1. First, the algorithm computes the MAs($n_t = sa0$) and MAs($n_t = sa1$), respectively. Then, it chooses an MA $n = v$ in MAs($n_t = sa0$) as n_{f1}. Based on the MA $n_{f1} = v$, the algorithm performs logic implications of $n_{f1} = v$ and MAs($n_t = sa1$) to obtain imp(($n_{f1} = v$) ∪ MAs($n_t = sa1$)). Finally, n_{f3} are the nodes that have the value of v' in MAs($n_t = sa1$)) and n_{f2} are the nodes that have different values in MAs($n_t = sa0$) and imp(($n_{f1} = v$) ∪ MAs($n_t = sa1$)). Therefore, the added substitute node n_a driven by n_{f1}, n_{f2}, and n_{f3} is Types 1–8 as shown in lines 4–9 of Algorithm 1. Similarly, in lines 10–15 of Algorithm 1, we can find the added substitute nodes of Types 9–16.

Here, we use the same example in Fig. 4 to demonstrate the proposed NAR algorithm. The detailed value assignments are also shown in Table I. Given a target node n_6, we want to find its added substitute node. First, we compute the MAs($n_6 = sa0$). To activate the fault-effect, the value of n_6 is set to 1. To propagate the fault-effect to the POs, the side-input pairs of the dominators are set to the corresponding values. Note that the dominator of n_6 is n_8, and we have to set the side-input pairs of the dominators to the noncontrolling pair enabling the propagation of the fault-effect, i.e., the value of n_7 has to be assigned to 1 for the fault-effect propagation. Then, we derive more MAs by performing logic implications forward and backward. Thus, $\{n_6 = 1,\ n_7 = 1,\ n_1 = 1,\ d = 1,\ n_2 = 1,\ e = 1,\ n_4 = 0,\ a = 1,\ b = 1,\ c = 1,\ n_3 = 1,\ n_5 = 1,\ n_8 = 1\}$ ∈ MAs($n_6 = sa0$). Second, we compute the MAs($n_6 = sa1$) in the same manner, and they are $\{n_6 = 0,\ n_7 = 1,\ n_2 = 1,\ n_4 = 0,\ e = 1,\ n_8 = 0\}$. Third, assume that we choose n_4 as n_{f3}, and n_3 as n_{f1}, we next compute imp(($n_3 = 1$) ∪ MAs($n_6 = sa1$)). The implication results are $\{n_6 = 0, n_7 = 1, n_2 = 1, n_4 = 0, e = 1, n_8 = 0,\ n_3 = 1,\ b = 1,\ c = 1,\ d = 1,\ n_1 = 0,\ a = 0,\ n_5 = 0\}$. As a result, one of a, n_1, and n_5 can be selected as n_{f2} because they all satisfy Conditions 2 and 3. If we select n_5 as n_{f2}, n_9 that is driven by n_3, n_5, and n_4 is an added substitute node for n_6 as shown in Fig. 4(c). n_9 will satisfy Condition 1, which means that n_9 can be viewed as n_s in Condition 1. Merging n_6 with n_9 will not affect the functionality of the circuit. Note that we do not consider n_8 as n_{f2}. The reason is that when selecting n_8 as n_{f2}, the replacement of n_6 with n_9 will result in a cycle, which is not allowed in this work.

TABLE I
VALUE ASSIGNMENTS OF THE PROPOSED NAR APPROACH PERFORMING ON THE EXAMPLE IN FIG. 4

MAs(n_6 = $sa0$)														
n_t	side-input	value assignments												
n_6	n_7	n_1	n_2	n_3	n_4	n_5	n_8	a	b	c	d	e		
1	1	1	1	1	0	1	1	1	1	1	1	1		
MAs(n_6 = $sa1$)														
n_t	side-input	value assignments												
n_6	n_7	n_1	n_2	n_3	n_4	n_5	n_8	a	b	c	d	e		
0	1		1		0		0					1		
(n_3 = 1) ∪ MAs(n_6 = $sa1$)														
n_t	side-input	value assignments												
n_6	n_7	n_1	n_2	n_3	n_4	n_5	n_8	a	b	c	d	e		
0	1		1	1	0		0					1		
***imp*((n_3 = 1) ∪ MAs(n_6 = $sa1$))**														
n_t	side-input	value assignments												
n_6	n_7	n_1	n_2	n_3	n_4	n_5	n_8	a	b	c	d	e		
0	1	0	1	1	0	0	0	0	1	1	1	1		

Algorithm 1: NAR Algorithm for Majority Circuits

```
1  Given a node $n_t$ in a majority circuit $C$.
2  Compute $MAs(n_t = sa0)$.
3  Compute $MAs(n_t = sa1)$.
4  for each MA $n = v$ in $MAs(n_t = sa0)$
5      (i) $n_{f3}\_set \leftarrow$ nodes that have the value of $v'$ in $MAs(n_t = sa1)$.
6      (ii) Take $n$ as $n_{f1}$.
7      (iii) Compute $imp((n_{f1} = v) \cup MAs(n_t = sa1))$.
8      (iv) $n_{f2}\_set \leftarrow$ nodes that have different values in $MAs(n_t = sa0)$ and $imp((n_{f1} = v) \cup MAs(n_t = sa1))$.
9  The set of $n_a$, which are driven by $n_{f1}$, $n_{f2}$, and $n_{f3}$, are Type 1$\sim$ Type 8.
10 for each MA $n = v$ in $MAs(n_t = sa1)$
11     (i) $n_{f3}\_set \leftarrow$ nodes that have the value of $v'$ in $MAs(n_t = sa0)$.
12     (ii) Take $n$ as $n_{f1}$.
13     (iii) Compute $imp((n_{f1} = v) \cup MAs(n_t = sa0))$.
14     (iv) $n_{f2}\_set \leftarrow$ nodes that have different values in $MAs(n_t = sa1)$ and $imp((n_{f1} = v) \cup MAs(n_t = sa0))$.
15 The set of $n_a$, which are driven by $n_{f1}$, $n_{f2}$, and $n_{f3}$, are Type 9 $\sim$ Type 16.
```

IV. MA Reuse and Circuit Size Reduction for Majority Circuits

In this section, we first present a technique to accelerate the MA computation in the NAR algorithm—MA reuse. Then we present our overall algorithm, including redundancy removal, node merging, and NAR techniques, for majority circuit size reduction.

A. MA Reuse

The MA computation plays an important role in finding added substitute nodes. If the MA computation can be accelerated, the NAR algorithm will be more efficient. We observed that the computed MAs of different fault tests could be identical. Chen and Wang [10], [11] proposed the MA reuse technique for AIGs. However, the MA reuse technique for majority circuits has not been considered yet. Hence, the technique of MA reuse for majority circuits is proposed in this work for obtaining MAs in a more efficient manner.

We use an example in Fig. 8(a) to demonstrate this idea. We assume that n_k has a fanin node n_i only driving n_k and a constant value of "0" as one fanin value. The fanout of n_k is not explicitly expressed in Fig. 8. First, we select n_k as the target node for computing the MAs($n_k = sa0$). n_k is set to "1" to activate the fault-effect and the side-input pairs of dominators of n_k are set to their corresponding input-noncontrolling values to propagate the fault-effect. For ease of discussion, we denote these assignments for propagating the fault-effect as P. After performing logic implications of $\{n_k = 1, P\}$, MAs($n_k = sa0$) are obtained as $\{n_k = 1, n_i = 1, n_j = 1, \text{imp}(P)\}$. Next, we consider the computation of MAs($n_i = sa0$). Similarly, to activate the fault-effect on n_i's stuck-at 0 fault, n_i is set to 1.

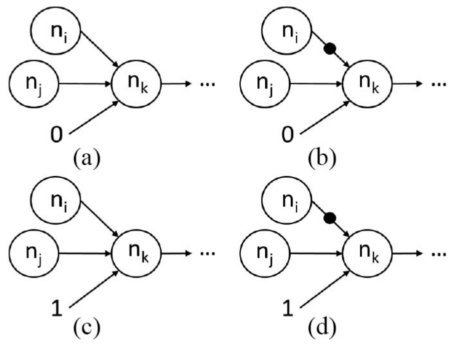

Fig. 8. Scenarios for MA reuse. (a) MAs($n_i = sa0$) = MAs($n_k = sa0$). (b) MAs($n_i = sa1$) = MAs($n_k = sa0$). (c) MAs($n_i = sa1$) = MAs($n_k = sa1$). (d) MAs($n_i = sa0$) = MAs($n_k = sa1$).

Because n_i only drives n_k, the dominators of n_i are the dominators of n_k and n_k itself. Thus, $\{n_j = 1, P\}$ are the assignments to propagate the fault-effect of n_i's stuck-at 0 fault. After performing implications of $\{n_i = 1, n_j = 1, P\}$, MAs($n_i = sa0$) are obtained as $\{n_k = 1, n_i = 1, n_j = 1, \text{imp}(P)\}$, which are the same as MAs($n_k = sa0$). Hence, when computing MAs($n_i = sa0$), we can reuse MAs($n_k = sa0$).

This idea can be applied in the circuit of Fig. 8(b), where MAs($n_i = sa1$) will be the same as MAs($n_k = sa0$). Furthermore, we can extend this idea to the circuit of Fig. 8(c), where n_k has a fanin node n_i only driving n_k and a constant value of "1" as one fanin value. Based on the similar derivations we presented, we can reuse MAs($n_k = sa1$) for computing MAs($n_i = sa1$). Again, in Fig. 8(d), MAs($n_i = sa0$) are identical to MAs($n_k = sa1$).

B. Circuit Size Reduction for Majority Circuits

1) Redundancy Removal: We know that the MAs are the unique and necessary value assignments for detecting a stuck-at fault on a node of circuit. If the computed MAs are inconsistent, the fault is untestable and the node is redundant. Specifically, in the NAR algorithm, we first compute the MAs($n_t = sa0$) and MAs($n_t = sa1$), respectively. If we find that the MAs($n_t = sa0$)/MAs($n_t = sa1$) are inconsistent, we can replace n_t with a constant value "0"/"1" and propagate this value "0"/"1" forward for simplifying circuits. This redundancy removal operation can be used to reduce the circuit size.

2) Node-Merging: As mentioned in Section II, using the node-merging approach for finding a substitute node n_s for a target node n_t requires the computations of MAs($n_t = sa0$) and MAs($n_t = sa1$) [17]. Since the NAR algorithm also requires these two sets of MAs as shown in Algorithm 1, we can integrate the node-merging approach with the NAR algorithm for majority circuit size reduction. Given a target node n_t, we first compute the MAs($n_t = sa0$) and MAs($n_t = sa1$). If we have found a substitute node n_s for n_t, we then directly apply the node-merging approach for circuit size reduction.

3) Overall Algorithm: The overall circuit size reduction algorithm for majority circuits is shown in Algorithm 2. Given a majority circuit C, each node in C is selected as n_t in the depth-first-search (DFS) order from the POs to PIs. We

Algorithm 2: Majority Circuit Size Reduction

```
   Input:  A given majority circuit $C$.
   Output: An optimized majority circuit $C_{\text{opt}}$.
1  for each node $n_t$ of $C$ in DFS order from POs to PIs
2      Compute $MAs(n_t = sa0)$ with MA reuse.
       // Redundancy removal
3      if ($MAs(n_t = sa0)$ is inconsistent) replace $n_t$ with 0
         and simplify the circuit, continue.
4      Compute $MAs(n_t = sa1)$ with MA reuse.
5      if ($MAs(n_t = sa1)$ is inconsistent) replace $n_t$ with 1
         and simplify the circuit, continue.
       // Node-merging for finding the $n_s$
6      $Substitute\_set$ ← nodes having different values in
         $MAs(n_t = sa0)$ and $MAs(n_t = sa1)$.
7      if ($Substitute\_set \neq \emptyset$) replace $n_t$ with a node in
         $Substitute\_set$, continue.
       // NAR for finding the $n_a$
8      if ($n_t$ has at least a fanin node only driving $n_t$)
9          for each MA $n = v$ in $MAs(n_t = sa0)$
10             (i) $n_{f3}\_set$ ← nodes having value of $v'$ in
                 $MAs(n_t = sa1)$.
11             (ii) Take $n$ as $n_{f1}$.
12             (iii) Compute
                 $imp((n_{f1} = v) \cup MAs(n_t = sa1))$.
13             (iv) $n_{f2}\_set$ ← nodes having different values
                 in $MAs(n_t = sa0)$ and
                 $imp((n_{f1} = v) \cup MAs(n_t = sa1))$.
14             (v) if ($n_{f3}\_set \neq \emptyset$ and $n_{f2}\_set \neq \emptyset$) replace
                 $n_t$ with a node driven by $n_{f1}$, $n_{f2}$, and $n_{f3}$,
                 break.
15         if ($n_t$ has been replaced) continue.
16         for each MA $n = v$ in $MAs(n_t = sa1)$
17             (i) $n_{f3}\_set$ ← nodes having value of $v'$ in
                 $MAs(n_t = sa0)$.
18             (ii) Take $n$ as $n_{f1}$.
19             (iii) Compute
                 $imp((n_{f1} = v) \cup MAs(n_t = sa0))$.
20             (iv) $n_{f2}\_set$ ← nodes having different values
                 in $MAs(n_t = sa1)$ and
                 $imp((n_{f1} = v) \cup MAs(n_t = sa0))$.
21             (v) if ($n_{f3}\_set \neq \emptyset$ and $n_{f2}\_set \neq \emptyset$) replace
                 $n_t$ with a node driven by $n_{f1}$, $n_{f2}$, and $n_{f3}$,
                 break.
```

compute the MAs($n_t = sa0$) and MAs($n_t = sa1$) with the MA reuse technique. If the MAs are inconsistent, the redundancy removal operation is applied. If the inconsistent MAs in MAs($n_t = sa0$) or MAs($n_t = sa1$) occur, we consider the next node in C after replacing n_t with the constant value "0" or "1," respectively. Otherwise, we search for the substitute node n_s for n_t. If we can find the nodes having different values in MAs($n_t = sa0$) and MAs($n_t = sa1$), these nodes are n_s for n_t. However, if n_t has no n_s for node merging, and n_t has at least one fanin node only driving n_t, the algorithm applies the NAR technique as shown in lines 8–21 of Algorithm 2 for circuit size reduction. After considering all the nodes in C as n_t for optimization, the optimized majority circuit C_{opt} is returned.

TABLE II
EXPERIMENTAL RESULTS OF REPLACEABLE NODE IDENTIFICATION ON WELL OPTIMIZED MIG BENCHMARKS [1]–[3] BY USING REIMPLEMENTED APPROACH [17] AND OUR APPROACH

Benchmark Infor.			**Re-im. [17]**		**Our Approach**	
Name	|PI|/|PO|	|Node|	Repl.(%)	Time(s)	Repl.(%)	Time(s)
usb_phy	113/111	372	5.9	0.3	8.1	0.4
ss_pcm	106/98	397	1.0	0.6	1.5	0.8
sasc	133/132	621	1.3	0.7	2.3	1.2
simple_spi	148/147	837	5.0	1.5	8.4	2.1
pci_spoci_ctrl	85/76	932	16.6	5.0	27.6	13.5
i2c	147/142	971	4.3	1.7	9.3	2.8
hamming	200/7	2071	9.7	6.5	14.6	7.4
sqrt32	32/16	2156	9.0	23.2	11.0	25.0
systemcdes	314/258	2453	10.3	20.5	12.9	40.5
spi	274/276	3337	7.1	55.9	7.5	78.2
des_area	368/72	4186	10.9	90.9	12.3	147.6
div16	32/32	4374	12.8	83.9	21.2	102.7
max	512/130	5210	1.3	114.2	1.5	126.7
mem_ctrl	1198/1225	7143	2.5	86.0	5.0	218.6
tv80	373/404	7397	5.7	202.1	9.2	494.1
revx	20/25	7516	12.4	155.5	19.0	178.4
MUL32	64/64	9096	11.2	63.8	15.4	85.7
MAC32	96/65	9326	5.1	37.1	5.9	37.0
systemcaes	930/819	9547	2.8	191.2	6.6	1019.3
ac97_ctrl	2255/2250	10745	3.0	57.1	4.2	94.6
usb_funct	1860/1846	12995	2.5	151.4	4.1	982.6
diffeq1	354/289	17725	7.8	239.5	10.7	367.4
square	64/127	17887	4.9	127.3	5.1	132.1
comp	279/193	18493	2.5	1280.5	4.5	1515.6
pci_bridge32	3519/3528	18603	1.3	315.4	2.2	1379.9
aes_core	789/668	20947	11.4	402.0	15.2	745.1
mult64	128/128	25772	3.85	366.7	4.0	465.45
DSP	4223/3953	40212	2.9	922.4	4.6	1308.7
Average	—	—	**6.3**	173.4	**9.1**	345.2
Ratio			**1**		**1.44**	

4) Time Complexity of the Overall Algorithm: The pseudocode of the overall circuit size reduction algorithm for majority circuits is shown in Algorithm 2. First, our algorithm goes through each node in the circuit in an outer for-loop as shown in lines 1–21 of Algorithm 2. If the node-merging approach fails to find a substitute node for the given target node n_t and n_t has at least one fanin node only driving n_t, we traverse the set of MAs for finding the n_a as shown in lines 9–14 and lines 16–21 of Algorithm 2. The complexity of our algorithm currently is $O(nm)$, where n is the number of nodes in the circuit and m represents the number of the MAs for an n_t. Second, the core of our algorithm is the logic implication for MA computation. The more MAs we can compute, the more nodes we can identify to replace the n_t. For each MA in the MA set, we next conduct logic implication in the inner for-loop in lines 12 and 19 of Algorithm 2. Assume that the complexity of one logic implication is $O(p)$. Then, the overall time complexity of our algorithm will be $O(nmp)$.

To reduce the time complexity of MA computation, we propose the MA reuse technique. However, it is not the case that the MA reuse technique is always applicable to one node. Hence, the MA reuse technique does not contribute to the complexity reduction to our algorithm. Nevertheless, the MA reuse technique does reduce the required CPU time as will be presented in the experimental results in Table III.

TABLE III
COMPARISON OF EXPERIMENTAL RESULTS ON CIRCUIT SIZE AND DEPTH ON WELL OPTIMIZED MIG BENCHMARKS [1]–[3] BETWEEN [17] AND OUR APPROACH

Benchmark Info.				**Re-im. [17]**			**MA.**	**Non-depth Preserved**			**MA.**	**Depth Preserved**			**MA.**
Name	|PI|/|PO|	|Node|	Depth	|Node|%	Depth%	Time	Redu.	|Node|%	Depth%	Time	Redu.	|Node|%	Depth%	Time	Redu.
usb_phy	106/98	372	7	2.69	0.00	0.22	18.52	3.49	0.00	0.29	14.71	3.23	0.00	0.20	16.67
ss_pcm	106/98	397	6	0.25	0.00	0.32	27.27	0.50	0.00	0.62	7.46	0.25	0.00	0.36	25.00
sasc	133/132	621	6	0.32	0.00	0.44	25.42	0.64	-16.67	0.68	28.42	0.32	0.00	0.52	22.39
simple_spi	148/147	837	8	0.96	0.00	0.98	21.60	2.63	-37.50	1.16	33.71	1.08	0.00	0.97	17.80
pci_spoci_ctrl	85/76	932	11	14.27	0.00	3.24	12.20	17.06	-45.45	4.76	31.81	14.59	0.00	3.80	6.63
i2c	147/142	971	8	1.96	0.00	1.17	24.03	3.40	-12.50	1.45	35.27	1.96	0.00	1.18	21.33
hamming	200/7	2071	61	5.70	0.00	5.5	5.45	8.98	-14.75	5.15	27.05	5.94	0.00	5.13	4.29
sqrt32	32/16	2156	164	3.90	0.00	18.10	11.32	4.22	-6.71	15.82	28.12	3.66	0.00	15.06	5.46
systemcdes	314/258	2453	19	3.30	0.00	18.60	14.64	4.40	-26.32	21.21	30.41	3.42	0.00	16.75	15.79
spi	274/276	3337	19	7.49	0.00	26.53	25.08	7.73	-15.79	30.88	27.93	7.52	0.00	26.12	20.70
des_area	368/72	4186	22	5.95	0.00	39.23	17.04	5.88	-13.64	47.10	7.97	5.66	0.00	38.86	12.14
max	512/130	4210	29	0.97	0.00	76.60	4.38	1.00	-3.45	72.54	33.17	0.95	0.00	67.30	14.06
div16	32/32	4374	102	7.73	0.00	75.80	3.75	10.54	-37.25	53.57	26.04	8.16	0.00	51.26	4.45
mem_ctrl	1198/1225	7143	19	1.23	0.00	53.82	11.09	2.00	-78.95	101.39	9.91	1.29	0.00	80.35	8.68
tv80	373/404	7397	30	2.81	0.00	117.39	14.40	4.11	-23.33	236.63	22.61	2.87	0.00	200.61	6.89
revx	20/25	7516	143	4.68	0.00	92.21	6.94	7.05	-34.97	114.13	25.90	4.83	0.00	111.02	4.48
MUL32	64/64	9096	36	5.02	0.00	43.04	9.84	6.89	-16.67	62.33	9.23	5.17	0.00	47.44	7.61
MAC32	96/65	9326	41	2.26	0.00	31.92	15.82	2.73	-41.46	36.89	20.17	2.26	0.00	33.97	2.92
systemcaes	930/819	9547	25	0.98	0.00	80.45	26.98	2.58	-64.00	429.82	5.70	0.97	0.00	303.49	3.54
ac97_ctrl	2255/2250	10745	8	0.15	0.00	25.61	22.56	0.74	0.00	48.15	28.41	0.15	0.00	39.37	13.98
usb_funct	1860/1846	12995	19	1.05	0.00	72.04	25.38	1.54	-31.58	414.49	21.62	1.01	0.00	200.52	10.82
diffeq1	354/289	17725	219	3.64	0.00	179.82	10.32	4.68	-20.09	237.32	29.37	3.78	0.00	242.75	5.32
square	64/127	17887	40	0.43	0.00	107.59	25.43	0.50	-25.00	117.15	3.17	0.43	0.00	114.88	2.58
comp	279/193	18493	77	1.64	0.00	635.82	18.23	1.91	-53.25	833.26	30.37	1.35	0.00	771.6	12.56
pci_bridge32	3519/3528	18603	16	0.58	0.00	146.65	15.93	0.92	-43.75	590.50	14.23	0.56	0.00	432.88	6.65
aes_core	789/668	20947	18	4.97	0.00	243.09	13.36	6.25	-61.11	372.14	12.35	5.67	0.00	323.45	11.71
mult64	128/128	25772	109	1.90	0.00	230.57	9.09	1.94	-4.59	275.49	14.03	1.92	0.00	242.19	9.83
DSP	4223/3953	40212	34	1.34	0.00	445.69	19.27	2.21	-44.12	678.72	14.49	1.37	0.00	545.83	14.51
Average (%)	—	—	—	**3.08**	0.00	—	**16.26**	**4.16**	-27.6	—	**21.20**	**3.23**	0.00	—	**11.03**
Ratio				**1**				**1.35**				**1.05**			

V. EXPERIMENTAL RESULTS

We implemented our NAR approach in C++ language, and conducted our experiments on an Intel Xeon E5-2650v2 2.60-GHz CentOS 6.7 platform with 64 GB. The MIG benchmarks were provided in [1], [2], and [3] and can be accessed from the EPFL Integrated Systems Laboratory [50], [51]. The XMG benchmarks we used were from EPFL Integrated Systems Laboratory [50], which were in the format of AIG. We then transformed the AIGs to XMGs by applying the methods proposed in [15] and [16] before the experiments.

We conducted four experiments. The first one is to present the different capabilities of replaceable node identification on MIGs between the node-merging approach [17] and our approach. The second one shows the circuit size reductions for MIGs by using the node-merging approach [17], Boolean resubstitution approach [34], [35], maj-n approach [31], cut-rewriting approach [38], and our approach. The third one shows the circuit size reductions for XMGs by using the node-merging approach [17] and our approach. Additionally, to show the applicability of our work, we also integrate our approach with the logic synthesis tool *CirKit* [49] and the EPFL logic synthesis libraries [38] for further optimization. The optimized circuits were also verified by using the logic equivalent checking tool—*cec* [47]. Additionally, since the quantum-dot cell cannot drive an arbitrary number of fanouts in practice, the number of fanouts for each node will be restricted to n, when running the proposed algorithm during QCA technology mapping [39]. We only performed our method when the number of fanouts is not exceeded. We conducted experiments for circuit size optimization by setting $n = 4$.

A. Replaceable Node Identification

A node is a *replaceable node* if and only if there exists a substitute node n_s or an added substitute node n_a for it. Given a target node, our approach first finds its substitute nodes. If no substitute nodes exist, it alternatively finds the added substitute nodes. In this experiment, the benchmarks are well-optimized circuits [1]–[3] and can be directly accessed online [51]. Each node in the circuit will be considered as a target node once. We compare the number of replaceable nodes in our approach against the reimplemented work [17], which only applies the node-merging technique. The experimental results are shown in Table II.

In Table II, Columns 1–3 lists the information of benchmarks, including names, the number of PIs and POs, and nodes. Columns 4–5 lists the results, including the percentage of the replaceable node count and the required CPU time measured in second, produced by the approach only using the node-merging approach [17]. Columns 6–7 lists the corresponding experimental results of our approach.

For example, the benchmark *aes_core* has 789 PIs and 668 POs, and 21522 nodes. 11.4% of nodes or 2387 nodes in the circuit are replaceable nodes using the node-merging approach. However, our approach can find the replaceable nodes for 15.2% of nodes or 3183 nodes in the circuit. According to Table II, the node-merging approach only found the replaceable nodes for 6.3% nodes in the benchmarks on average.

TABLE IV
COMPARISON OF EXPERIMENTAL RESULTS ON CIRCUIT SIZE AND DEPTH ON EPFL MIG BENCHMARKS [50] BETWEEN THE BOOLEAN RESUBSTITUTION [34] AND OUR APPROACH

Benchmark Info.				**Boolean Resub. [34]**			**Non-depth Preserved**			**Depth Preserved**		
Name	|PI|/|PO|	|Node|	Depth	|Node|%	Depth%	Time	|Node|%	Depth%	Time	|Node|%	Depth%	Time
ctrl	7/26	174	10	45.98	-10.00	2.08	31.61	-20.00	0.29	25.29	0.00	0.29
router	60/30	257	54	0.00	0.00	0.25	47.08	0.00	0.21	47.08	0.00	0.24
int2float	11/7	260	16	10.00	0.00	2.45	16.92	-100.00	0.68	12.31	0.00	0.60
dec	8/256	304	3	0.00	0.00	0.30	0.00	0.00	0.55	0.00	0.00	0.83
cavlc	10/11	693	16	10.53	0.00	42.47	7.22	-81.25	7.39	3.17	0.00	6.27
priority	128/8	978	250	20.65	24.80	0.78	6.34	0.00	1.33	6.34	0.00	1.11
adder	256/129	1020	255	12.45	0.39	0.75	12.35	0.00	0.97	12.35	0.00	0.87
i2c	147/142	1342	20	4.62	0.00	1.79	16.54	-15.00	3.36	10.58	0.00	3.34
max	512/130	2865	287	0.87	0.00	1.62	0.00	0.00	22.52	0.00	0.00	21.24
bar	135/128	3336	12	7.91	0.00	3.09	0.00	0.00	16.09	0.00	0.00	16.89
sin	24/25	5416	225	1.42	-0.44	10.58	0.66	0.00	250.07	0.66	0.00	253.50
voter	1001/1	13758	70	14.77	-1.43	19.25	22.76	0.00	45.37	22.76	0.00	37.44
square	64/128	18484	250	1.32	-0.40	25.32	0.80	0.00	130.40	0.80	0.00	128.00
sqrt	128/128	24618	5058	11.03	-36.16	28.55	15.17	-0.06	1413.20	5.36	0.00	1401.82
div	128/128	57247	2505	1.04	-0.82	78.88	0.67	0.00	1918.20	0.67	0.00	1897.15
Average (%)	—	—	—	**9.51**	-1.60	—	**11.88**	-14.42	—	**9.83**	0.00	—
Ratio				**1**			**1.25**					

While our approach found 9.1% replaceable nodes in the benchmarks on average with the CPU time of 345.2 s.

In summary, our approach can find replaceable nodes for 44% more target nodes with the CPU time overhead of only 171.8 s on average.

B. Circuit Size Reduction

In this experiment, we implemented two versions of our approach: 1) for nondepth preservation, which focuses on optimizing the circuit size allowing depth overhead if any and 2) for depth preservation, which optimizes the circuit size without changing circuit's depth. We compare the circuit size reduction using our approach against the work [17] at first. The experiments were conducted on well-optimized circuits represented in MIGs [1]–[3], and the results are as shown in Table III. Note that the initial node count in each benchmark is slightly different from the results shown in [17]. The reason is that we do not have the original MIG benchmarks as listed in [17]. Presently, the MIG benchmarks we have were downloaded from the website [51] and Table III shows their information. To confirm the correctness of the benchmark information, we also use another synthesis tool *CirKit* [49] to obtain the information. The results show that the benchmark information in Table III are matched correctly.

In Table III, Columns 1–4 lists the information of benchmarks, including names, the number of PIs and POs, nodes, and depth. Columns 5–8 list the experimental results produced by the reimplemented node-merging approach [17], including the percentage of size reduction, the percentage of depth reduction, the required CPU time, and the percentage of the required CPU time reduction by using the MA reuse technique. Columns 9–12 and 13–16 lists the corresponding results of our nondepth preserved and depth preserved versions, respectively. For a fair comparison, we only applied the node-merging approach [17] and our approach once on the MIG benchmarks.

According to Table III, our approach achieved more circuit size reduction with a ratio of 1.35 as compared with the approach that only uses node-merging technique [17]. The MA reuse technique also decreases the required CPU time for the approaches by 16.26%, 21.20%, and 11.03%, respectively, on average.

Then, we compare our approach against the Boolean resubstitution method as shown in Table IV. The maximum number of PIs in a window of Boolean resubstitution is set to 6. The benchmarks we used were from the EPFL Integrated Systems Laboratory [50]. In Table IV, Columns 1–4 lists the information of benchmarks. Columns 5–7 list the results produced by the Boolean resubstitution approach [34], [35], including the percentage of size reduction, the percentage of depth reduction, and the required CPU time measured in second. Columns 8–10 and 11–13 lists the corresponding results of our nondepth preserved and depth preserved versions, respectively. For a fair comparison, we only applied our method and the Boolean resubstitution method once on the EPFL benchmarks and evaluated the resultant circuits produced by these two methods.

According to Table IV, our nondepth preserved approach achieved more circuit size reduction with a ratio of 1.25 as compared with the Boolean resubstitution approach. Additionally, our depth preserved approach can reduce the circuit node count by 9.45% on average without increasing the depth. The Boolean resubstitution approach can optimize the circuit more efficiently but increase the depth of the circuit.

To show the optimization capability of the proposed approach, we also compare our approach against other size optimization methods for MIGs—cut-rewriting [38], and maj-*n* [31] approaches. The cut size of the cut-rewriting approach is set to 4, and the maximum number of inputs is limited to 3 in the maj-*n* approach. The experimental results are shown in Table V. Note that for a fair comparison, we only applied the cut-rewriting approach by "cut_rewrite" implemented in the EPFL libraries [38], the maj-*n* approach by "&if -M 3" proposed by Neutzling *et al.* [31], and our two versions of the proposed approach for once on the benchmarks.

The MIG benchmarks we used are the same as listed in Table IV. In Table V, Columns 5–7 lists the results produced

TABLE V
COMPARISON OF EXPERIMENTAL RESULTS ON CIRCUIT SIZE AND DEPTH ON EPFL MIG BENCHMARKS [50] AMONG CUT-REWRITING APPROACH [38], MAJ-N APPROACH [31], AND OUR APPROACH

Benchmark Info.				**cut-rewriting [38]**			**maj-n [31]**			**Non-depth Preserved**			**Depth Preserved**		
Name	|PI|/|PO|	|Node|	Depth	|Node|%	Depth%	Time	|Node|%	Depth%	Time	|Node|%	Depth%	Time	|Node|%	Depth%	Time
ctrl	7/26	174	10	20.69	0.00	0.54	13.22	0.00	0.13	31.61	-20.00	0.29	25.29	0.00	0.29
router	60/30	257	54	4.67	0.00	1.17	19.84	0.00	0.15	47.08	0.00	0.21	47.08	0.00	0.24
int2float	11/7	260	16	11.92	6.25	0.73	13.46	0.00	0.15	16.92	-100.00	0.68	12.31	0.00	0.60
dec	8/256	304	3	0.00	-33.33	0.05	0.00	0.00	0.13	0.00	0.00	0.55	0.00	0.00	0.83
cavlc	10/11	693	16	4.91	0.00	2.17	-1.30	0.00	0.18	7.22	-81.25	7.39	3.17	0.00	6.27
priority	128/8	978	250	4.91	1.60	3.04	25.87	0.40	0.25	6.34	0.00	1.33	6.34	0.00	1.11
adder	256/129	1020	255	12.45	-0.39	1.84	12.45	49.41	0.22	12.35	0.00	0.97	12.35	0.00	0.87
i2c	147/142	1342	20	5.96	10.00	3.99	0.15	0.00	0.26	16.54	-15.00	3.36	10.58	0.00	3.34
max	512/130	2865	287	15.67	26.83	11.19	11.59	25.78	0.87	0.00	0.00	22.52	0.00	0.00	21.24
bar	135/128	3336	12	7.70	-8.33	12.60	0.00	0.00	0.45	0.00	0.00	16.09	0.00	0.00	16.89
sin	24/25	5416	225	5.30	13.78	23.32	-3.29	8.44	2.07	0.66	0.00	250.07	0.66	0.00	253.50
voter	1001/1	13758	70	23.36	1.43	66.46	8.71	0.00	3.78	22.76	0.00	45.37	22.76	0.00	37.44
square	64/128	18484	250	2.35	0.40	73.90	4.36	34.80	3.54	0.80	0.00	130.40	0.80	0.00	128.00
sqrt	128/128	24618	5058	22.31	-18.37	272.76	0.00	0.02	4.67	15.17	-0.06	1413.20	5.36	0.00	1401.82
div	128/128	57247	2505	31.34	-75.61	431.56	0.06	0.11	9.55	0.67	0.00	1918.20	0.67	0.00	1897.15
Average (%)	—	—	—	**10.57**	-5.05	—	**7.01**	8.03	—	**11.88**	-14.42	—	**9.83**	0.00	—
Ratio				**1**						**1.12**					
Ratio							**1**			**1.69**					

TABLE VI
COMPARISON OF EXPERIMENTAL RESULTS ON CIRCUIT SIZE AND DEPTH ON WELL-OPTIMIZED XMG BENCHMARKS [15], [16] BETWEEN [17] AND OUR APPROACH

Benchmark Info.				**Re-im. [17]**			**Non-depth Preserved**			**Depth Preserved**		
Name	|PI|/|PO|	|Node|	Depth	|Node|%	Depth	Time	|Node|%	Depth	Time	|Node|%	Depth	Time
ctrl	7/26	133	7	12.78	14.29	0.10	14.29	0.00	0.14	14.29	0.00	0.13
int2float	60/30	245	23	10.61	0.00	0.19	11.43	-8.70	0.20	10.61	0.00	0.20
router	11/7	250	13	8.00	0.00	0.72	11.60	0.00	0.79	9.60	0.00	0.83
dec	8/256	304	3	0.00	0.00	0.14	0.00	0.00	0.14	0.00	0.00	0.14
adder	256/129	383	128	0.00	0.00	0.04	0.00	0.00	0.03	0.00	0.00	0.04
cavlc	10/11	716	16	7.82	0.00	6.83	11.87	-50.00	7.71	10.06	0.00	7.68
priority	128/8	915	86	2.19	0.00	1.39	10.16	0.00	1.52	9.62	0.00	1.42
i2c	147/142	1314	16	6.32	6.25	2.61	17.72	-25.00	3.22	13.69	0.00	3.32
max	512/130	2151	238	2.42	0.00	80.41	2.42	0.00	81.25	2.42	0.00	84.22
bar	135/128	2965	13	5.53	0.00	11.39	5.53	0.00	11.50	5.53	0.00	11.81
sin	24/25	3893	130	1.64	3.85	83.59	2.26	-8.46	94.88	1.93	0.00	96.83
voter	1001/1	6885	56	17.37	3.57	17.27	22.24	0.00	17.33	19.42	0.00	18.2
square	64/128	12760	127	4.26	0.00	140.36	5.31	0.00	148.22	5.27	0.00	149.41
multiplier	128/128	15924	134	0.36	0.00	130.9	0.36	0.00	136.34	0.36	0.00	132.54
sqrt	128/128	17344	4260	2.25	0.00	502.38	2.76	0.00	552.35	2.25	0.00	596.42
div	128/128	42812	2505	4.57	1.20	1695.99	5.10	0.00	2151.00	4.79	0.00	2100.93
Average (%)	—	—	—	**5.38**	1.82	—	**7.69**	-5.76	—	**6.92**	0.00	—
Ratio				**1**			**1.43**			**1.29**		

by the cut-rewriting approach [38], including the percentage of size reduction, the percentage of depth reduction, and the required CPU time measured in second. Columns 8–10, 11–13, and 14–16 lists the corresponding results of maj-*n*, our nondepth preserved, and depth preserved versions, respectively.

According to Table V, our nondepth preserved version achieved more circuit size reduction with a ratio of 1.12 and 1.69 as compared with the cut-rewriting approach and the maj-*n* approach, respectively. For the depth preserved version, our result is still competitive.

After that, we conducted the experiment of circuit size reduction on XMGs. The XMG benchmarks we used were also from EPFL Integrated Systems Laboratory [50], which were originally provided in the format of AIG. We transformed the AIGs to XMGs before the experiments by applying the methods proposed in [15] and [16].

Similarly, the corresponding experimental results for XMGs are summarized in Table VI. Our two versions of approaches achieved more circuit size reduction with a ratio of 1.43 and 1.29 as compared with the approach that only uses node-merging technique [17], respectively. Note that our approach focuses on optimizing MAJ nodes from the circuits. For XOR nodes in XMGs, we do not deal with them. This is because XOR nodes do not have controlling and noncontrolling values. As mentioned in Section II, the MAs are necessary values assigned to some nodes in the circuits for detecting a fault on a node. However, detecting a fault on an XOR node will not generate any necessary assignments on the XOR node, which means that Conditions 1–3 cannot be applied on XOR nodes. Thus, our approach only optimizes the MAJ nodes in the XMGs.

In the last experiment, we integrate the state-of-the-art logic synthesis tool for majority logic—*CirKit* [49] and the EPFL logic synthesis library [38]—with the approach that only uses the node-merging technique [17], and our approach, respectively, to present the effectiveness of our approach for circuit optimization. Here, we use the original MIG benchmarks [1],

TABLE VII
COMPARISON OF EXPERIMENTAL RESULTS AMONG *mig_resub.* × 6, ([17] + *mig_resub.*) × 3, AND (*Ours* + *mig_resub.*) × 3

Benchmark Info.				***mig_resub.*×6**			**([17]+*mig_resub.*)×3**			**(*Ours*+*mig_resub.*)×3**		
Name	|PI|/|PO|	|Node|	Depth	|Node|%	Depth%	Time	|Node|%	Depth%	Time	|Node|%	Depth%	Time
ss_pcm	106/98	405	7	0.74	0.00	2.58	0.74	0.00	2.14	0.74	0.00	2.34
usb_phy	113/111	460	10	6.74	0.00	7.24	11.74	10.00	2.81	12.39	10.00	2.93
sasc	133/132	773	9	0.78	0.00	12.31	1.16	0.00	5.08	1.68	11.11	6.90
simple_spi	148/147	1053	12	2.28	-8.33	10.92	2.85	-8.33	7.4	5.22	-75.00	11.86
sqrt32	32/16	1113	495	0.45	0.61	4.82	0.54	0.61	7.60	0.54	0.61	8.71
i2c	147/142	1166	14	3.77	-7.14	9.38	8.66	0.00	7.06	13.04	-14.29	9.00
pci_spoci_ctrl	85/76	1386	18	12.70	-5.56	8.79	30.01	-5.56	11.9	33.41	-38.89	18.99
max	512/130	2865	287	0.87	0.00	10.89	0.87	0.00	53.69	0.87	0.00	59.96
systemcdes	314/258	2999	27	9.34	-25.93	16.55	14.94	-29.63	240.86	15.64	-37.04	256.81
hamming	200/7	3612	75	6.76	-16.00	17.06	15.01	-20.00	33.736	28.27	-33.33	45.41
spi	274/276	3808	32	1.71	0.00	30.46	2.65	0.00	62.19	3.28	-6.25	160.02
des_area	368/72	4857	33	16.29	-9.09	21.99	17.79	-21.21	348.76	18.32	-9.09	373.02
div16	32/32	7175	136	6.75	-13.24	34.76	13.07	-16.91	209.14	28.11	-63.97	272.05
tv80	373/404	9647	52	4.20	-1.92	72.27	9.96	-1.92	404.70	12.35	-9.62	968.08
MUL32	64/64	11613	43	6.54	-16.28	64.54	17.76	-13.95	156.12	21.91	-79.07	253.32
MAC32	96/65	12062	69	6.54	-8.70	75.59	9.42	-10.14	128.8	11.72	-13.04	153.82
systemcaes	930/819	12384	46	1.57	2.17	94.94	3.55	2.17	388.41	4.66	2.17	2819.99
ac97_ctrl	2255/2250	14268	12	0.92	0.00	187.52	1.45	0.00	155.66	1.54	0.00	255.08
mem_ctrl	1198/1225	15337	36	4.04	5.56	77.65	48.26	5.56	274.72	50.32	19.44	546.17
usb_funct	1860/1846	15894	27	3.98	3.70	119.60	6.26	3.70	183.87	8.07	-3.70	675.03
revx	20/25	16164	192	24.47	-5.21	90.28	38.39	0.52	532.48	48.89	-29.17	717.08
aes_core	789/668	21522	26	4.27	0.00	172.84	5.15	0.00	451.62	6.87	-34.62	1153.51
pci_bridge32	3519/3528	22806	30	0.84	-3.33	259.29	1.21	0.00	389.03	1.84	0.00	3648.46
diffeq1	354/289	33536	303	31.82	-1.98	251.70	33.68	-3.96	892.21	40.54	-4.62	1225.20
DSP	4223/3953	45420	63	3.44	-9.52	378.53	4.65	-7.94	1591.32	6.40	-1.59	3367.05
Average	—	—	—	**6.47**	-4.81	81.30	**11.99**	-4.68	261.66	**15.06**	-16.40	680.43
Ratio				**1**			**1.85**			**2.32**		
Ratio							**1**			**1.26**		

TABLE VIII
COMPARISON OF EXPERIMENTAL RESULTS AMONG *cut_rewrite* × 6, ([17] + *cut_rewrite*) × 3, AND (*Ours* + +*cut_rewrite*) × 3

Benchmark Info.				***cut_rewrite*×6**			**([17]+*cut_rewrite*)×3**			**(*Ours*+*cut_rewrite*)×3**		
Name	|PI|/|PO|	|Node|	Depth	|Node|%	Depth%	Time	|Node|%	Depth%	Time	|Node|%	Depth%	Time
ss_pcm	106/98	405	7	2.22	0.00	0.93	2.22	0.00	1.58	15.06	0.00	1.87
usb_phy	113/111	460	10	10.87	0.00	1.00	15.87	10.00	1.79	26.30	20.00	2.48
sasc	133/132	773	9	0.00	-22.22	1.66	4.14	-22.22	2.93	31.44	-11.11	3.80
simple_spi	148/147	1053	12	2.75	0.00	2.28	4.08	0.00	7.16	29.34	0.00	6.46
sqrt32	32/16	1113	495	3.86	5.66	2.03	3.86	5.66	7.73	5.48	6.87	12.54
i2c	147/142	1166	14	4.12	-7.14	2.38	7.29	0.00	7.54	25.04	7.14	7.74
pci_spoci_ctrl	85/76	1386	18	9.02	0.00	2.94	30.66	0.00	13.27	40.12	5.56	17.12
max	512/130	2865	287	12.67	26.13	5.85	12.67	26.13	310.60	15.67	26.83	305.70
systemcdes	314/258	2999	27	5.74	-3.70	7.12	6.90	-3.70	31.62	20.84	-7.41	47.24
hamming	200/7	3612	75	9.08	-12.00	7.04	15.73	-8.00	43.45	24.89	-933	47.93
spi	274/276	3808	32	1.97	3.13	9.15	2.44	3.13	97.88	3.18	3.13	130.36
des_area	368/72	4857	33	1.11	0.00	11.71	1.34	0.00	136.04	2.08	0.00	144.01
div16	32/32	7175	136	17.52	-15.44	14.49	24.43	-18.38	298.80	31.62	-21.32	312.67
tv80	373/404	9647	52	4.26	0.00	23.03	9.97	3.85	644.91	11.59	3.85	909.81
MUL32	64/64	11613	43	15.72	-11.63	40.29	17.85	-9.30	191.04	26.20	-9.30	196.01
MAC32	96/65	12062	69	35.19	11.59	48.10	32.38	0.00	161.11	34.21	2.90	154.73
systemcaes	930/819	12384	46	6.92	0.00	50.10	8.24	0.00	1131.26	8.45	0.00	2441.21
ac97_ctrl	2255/2250	14268	12	1.61	0.00	65.50	2.39	0.00	261.85	2.51	0.00	499.88
mem_ctrl	1198/1225	15337	36	7.18	0.00	67.50	51.83	8.33	656.81	53.37	8.33	765.53
usb_funct	1860/1846	15894	27	7.06	-3.70	79.20	9.05	-3.70	339.04	11.06	-3.70	462.69
revx	20/25	16164	192	10.93	6.25	1275.43	41.39	4.69	1033.84	46.88	5.73	1141.14
aes_core	789/668	21522	26	6.58	-7.69	271.00	7.14	-7.69	791.31	8.30	-7.69	970.61
pci_bridge32	3519/3528	22806	30	2.79	0.00	268.50	3.40	3.33	1312.57	4.09	3.33	1366.12
diffeq1	354/289	33536	303	28.29	-5.61	545.10	30.98	-3.63	2443.94	35.78	-2.97	2741.59
DSP	4223/3953	45420	63	5.26	-14.29	360.90	6.68	-12.70	2041.53	7.87	-12.70	2801.08
Average	—	—	—	**8.51**	-2.53	126.53	**14.12**	-0.97	478.78	**20.85**	0.32	619.61
Ratio				**1**			**1.66**			**2.45**		
Ratio							**1**			**1.48**		

which can be directly accessed online [51]. The experimental results are summarized in Table VII. The synthesis script that is integrated with our approach is "mig_resubstitution," which has been integrated in the synthesis tool *CirKit* and the EPFL logic synthesis library. Columns 5–7 show the results after running *mig_resubstitution* script for six times—*mig_resub.* × 6. Columns 8–10 list the results produced by the approach that only uses the node-merging technique [17] followed by *mig_resubstitution* script for three times iteratively—([17] + *mig_resub.*) × 3. Columns 11–13 list the results produced by our approach followed by *mig_resubstitution* script for three times iteratively—(*Ours* + *mig_resub.*) × 3. According to Table VII, the integration of our approach and *mig_resubstitution* achieved more size reduction with a ratio of 1.26 as compared to the integration of the approach that only uses the node-merging technique [17] and *mig_resubstitution*. As compared to *mig_resubstitution* only, the circuit size reduction by using our approach is even higher. Additionally, we also integrate our approach with the cut-rewriting approach, which also has been integrated in the EPFL logic synthesis library as a synthesis script—"cut_rewrite," based on the same experimental scheme. The corresponding experimental results are as shown in Table VIII. According to Table VIII, the integration of our approach and *cut_rewrite* can achieve more size reduction with a ratio of 1.48 as compared to the integration of the approach that only uses the node-merging technique [17] and *cut_rewrite*. From these experimental results, we realize that our approach can be well integrated with the state-of-the-art tool and libraries to achieve more circuit size reduction on majority circuits.

VI. Conclusion

In this article, we propose an NAR approach for majority circuit optimization. The NAR approach adds a new node in the circuit for replacing the given target node. We also propose an MA reuse technique for accelerating the added substitute node identification. The integration of our approach, the state-of-the-art synthesis tool *CirKit*, and the EPFL logic synthesis libraries significantly elevates the optimization capability for majority circuits. Scalability is a crucial problem in the proposed NAR approach. Presently, the proposed NAR approach only focuses on reducing the node number in a circuit as many as possible without considering runtime overhead. Additionally, if the NAR approach can be extended to adding more than one node, the opportunities for circuit restructuring and size reduction would increase. Our future work is to propose an approach to balance the circuit size reduction and runtime overhead as well as consider multiple node addition.

Acknowledgment

The authors would like to thank the EPFL LSI research group led by Prof. Giovanni De Micheli for providing the EPFL libraries and benchmarks, and the research group led by Prof. Renato Perez Ribas for providing the program of maj-*n*. They also thank Dr. Riener and Dr. Neutzling for their quick and fruitful responses to our questions about the programs.

References

[1] L. Amarú, P. E. Gaillardon, and G. D. Micheli, "Majority-inverter graph: A novel data-structure and algorithm for efficient logic optimization," in *Proc. DAC*, 2014, pp. 1–6.

[2] L. Amarú, P. E. Gaillardon, and G. D. Micheli, "Majority-inverter graph: A new paradigm for logic optimization," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 35, no. 5, pp. 806–819, May 2016.

[3] L. Amaru, P. E. Gaillardon, and G. D. Micheli, "Boolean logic optimization in majority-inverter graphs," in *Proc. DAC*, 2015, pp. 1–6.

[4] L. Amaru, P. E. Gaillardon, and G. D. Micheli, "Majority-based synthesis for nanotechnologies," in *Proc. ASP-DAC*, 2016, pp. 499–502.

[5] A. N. Bahar, S. Waheed, N. Hossain, and M. Asaduzzaman, "A novel 3-input XOR function implementation in quantum-dot cellular automata with energy dissipation analysis," *Alexandria Eng. J.*, vol. 57, no. 2, pp. 729–738, Jun. 2018.

[6] S.-C. Chang, L. P. P. P. van Ginneken, and M. Marek-Sadowska, "Fast boolean optimization by rewiring," in *Proc. ICCAD*, 1996, pp. 262–269.

[7] S.-C. Chang, M. Marek-Sadowska, and K. T. Cheng, "Perturb and simplify: Multilevel Boolean network optimizer," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 15, no. 12, pp. 1494–1504, Dec. 1996.

[8] S.-C. Chang, K.-T. Cheng, N.-S. Woo, and M. Marek-Sadowska, "Postlayout logic restructuring using alternative wires," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 16, no. 6, pp. 587–596, Jun. 1997.

[9] Y.-C. Chen and C.-Y. Wang, "An improved approach for alternative wires identification," in *Proc. ICCD*, 2005, pp. 711–716.

[10] Y.-C. Chen and C.-Y. Wang, "Fast detection of node mergers using logic implications," in *Proc. ICCAD*, 2009, pp. 785–788.

[11] Y.-C. Chen and C.-Y. Wang, "Fast node merging with don't cares using logic implications," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 29, no. 11, pp. 1827–1832, Nov. 2010.

[12] Y.-C. Chen and C.-Y. Wang, "Node addition and removal in the presence of don't cares," in *Proc. DAC*, 2010, pp. 505–510.

[13] Y.-C. Chen and C.-Y. Wang, "Logic restructuring using node addition and removal," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 32, no. 2, pp. 260–270, Feb. 2012.

[14] K. T. Cheng and L. A. Entrena, "Multi-level logic optimization by redundancy addition and removal," in *Proc. DAC*, 1993, pp. 373–377.

[15] Z. Chu, M. Soeken, Y. Xia, L. Wang, and G. D. Micheli, "Advanced functional decomposition using majority and its applications," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 39, no. 8, pp. 1621–1634, Aug. 2020.

[16] Z. Chu, L. Shi, L. Wang, and Y. Xia, "Multi-objective algebraic rewriting in XOR-majority graphs," *Integr. VLSI J.*, vol. 69, pp. 40–49, Nov. 2019.

[17] C.-C. Chung, Y.-C. Chen, C.-Y. Wang, and C.-C. Wu, "Majority logic circuits optimisation by node merging," in *Proc. ASP-DAC*, 2017, pp. 714–719.

[18] L. A. Entrena and K.-T. Cheng, "Combinational and sequential logic optimization by redundancy addition and removal," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 14, no. 7, pp. 909–916, Jul. 1995.

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

[20] W. Haaswijk, M. Soeken, L. Amaru, P. E. Gaillardon, and G. D. Micheli, "LUT mapping and optimization for majority-inverter graph," in *Proc. IWLS*, 2016.

[21] W. Haaswijk, M. Soeken, L. Amaru, P. E. Gaillardon, and G. D. Micheli, "A Novel Basis for Logic Rewriting," in *Proc. ASP-DAC*, 2017, pp. 151–156.

[22] L. Hellerman, "A catalog of three-variable or-invert and and-invert logical circuits," *IEEE Trans. Electron. Comput.*, vol. EC-12, no. 3, pp. 198–223, Jun. 1963.

[23] M. B. Khosroshahy, M. H. Moaiyeri, K. Navi, and N. Bagherzadeh, "An energy and cost efficient majority-based RAM cell in quantum-dot cellular automata," *Results Phys.*, vol. 7, pp. 3543–3551, 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2211379717306642

[24] T. Kirkland and M. R. Mercer, "A topological search algorithm for ATPG," in *Proc. DAC*, 1987, pp. 502–508.

[25] A. Kuehlmann, "Dynamic transition relation simplification for bounded propery checking," in *Proc. ICCAD*, 2004, pp. 50–57.

[26] K. Kong, Y. Shang, and R. Lu, "An optimized majority logic synthesis methodology for quantum-dot cellular automata," *IEEE Trans. Nanotechnol.*, vol. 9, no. 2, pp. 170–183, Mar. 2010.

[27] W. Kunz and D. K. Pradhan, "Recursive learning: A new implication technique for efficient solutions to CAD problems-test, verification, and optimization," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 13, no. 9, pp. 1143–1158, Sep. 1994.

[28] C. S. Lent, P. D. Tougaw, and W. Porod, "Bistable saturation in coupled quantum dots for quantum cellular automata," *Appl. Phys. Lett.*, vol. 62, no. 7, pp. 714–716, 1993.

[29] C. S. Lent and P. D. Tougaw, "A device architecture for computing with quantum dots," in *Proc. IEEE*, 1997, pp. 541–557.

[30] C.-C. Lin and C.-Y. Wang, "Rewiring using IRredundancy removal and addition," in *Proc. DATE*, 2009, pp. 324–327.

[31] A. Neutzling, F. S. Marranghello, J. M. Matos, A. Reis, and R. P. Ribas, "Maj-n logic synthesis for emerging technology," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 39, no. 3, pp. 747–751, Mar. 2020.

[32] S. Plaza, K. H. Chang, I. Markov, and V. Bertacco, "Node mergers in the presence of don't cares," in *Proc. ASP-DAC*, 2007, pp. 414–419.

[33] A. A. Prager, A. O. Orlov, and G. L. Snider, "Integration of CMOS, single electron transistors, and quantumdot cellular automata," in *Proc. Nanotechnol. Mater. Devices Conf.*, 2009, pp. 54–58.

[34] H. Riener, E. Testa, L. Amaru, M. Soeken, and G. D. Micheli, "Size optimization of MIGs with an application to QCA and STMG technologies," in *Proc. Int. Symp. Nanoscale Archit. (NANOARCH)*, 2018, pp. 157–162.

[35] H. Riener *et al.*, "Scalable generic logic synthesis: One approach to rule them all," in *Proc. DAC*, 2019, p. 70.

[36] M. Soeken, L. Amaru, P. E. Gaillardon, and G. D. Micheli, "Optimizing majority-inverter graphs with functional hashing," in *Proc. DATE*, 2016, pp. 1030–1035.

[37] M. Soeken, L. Amaru, P. E. Gaillardon, and G. D. Micheli, "Exact synthesis of majority-inverter graphs and its applications," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 36, no. 11, pp. 1842–1855, Nov. 2017.

[38] M. Soeken, H. Riener, W. Haaswijk, and G. D. Micheli, "The EPFL logic synthesis librarie," 2018. [Online]. Available: arXiv:1805.05121

[39] E. Testa *et al.*, "Inverter propagation and fan-out constraints for beyond-CMOS majority-based technologies," in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI*, 2017, pp. 164–169.

[40] J. Timler and C. S. Lent, "Power gain and dissipation in quantum-dot cellular automata," *J. Appl. Phys.*, vol. 91, no. 2, pp. 823–831, 2002.

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

[42] R. L. Wigington, "A new concept in computing," in *Proc. Inst. Radio Eng.*, vol. 47, no. 4, pp. 516–523, Apr. 1959.

[43] M. Wilson, K. Kannangara, G. Smith, M. Simmons, and B. Raguse, *Nanotechnology: Basic Science and Emerging Technologies*. Boca Raton, FL, USA: Chapman & Hall/Chemical Rubber Company, 2002.

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

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

[46] Q. Zhu, N. Kitchen, A. Kuehlmann, and A. Sangiovanni-Vincentelli, "SAT sweeping with local observability don't cares," in *Proc. DAC*, 2006, pp. 229–234.

[47] *ABC Package*. Accessed: Oct. 20, 2020. [Online]. Available: https://people.eecs.berkeley.edu/alanmi/abc/

[48] *ALSO*. Accessed: Oct. 20, 2020. [Online]. Available: https://github.com/nbulsi/also

[49] *CirKit*. Accessed: Oct. 20, 2020. [Online]. Available: https://msoeken.github.io/cirkit.html

[50] *EPFL Benchmarks*. Accessed: Oct. 20, 2020. [Online]. Available: epfl.ch/labs/lsi/downloads/.

[51] *MIG Benchmarks*. Accessed: Oct. 20, 2020. [Online]. Available: epfl.ch/labs/lsi/page-102566-en-html/mig/

Chang-Cheng Ko received the B.S. degree from the Department of Computer Science, National Chung Cheng University, Chiayi, Taiwan, in 2018. He is currently pursuing the M.S. degree with the Department of Computer Science, National Tsing Hua University, Hsinchu, Taiwan.

His current research interests include logic synthesis, optimization, verification for very large-scale integration designs, and automation for emerging technologies.

Chia-Chun Lin received the B.S. and M.S. degrees from the Department of Computer Science, National Tsing Hua University, Hsinchu, Taiwan, in 2011 and 2013, respectively, where he is currently pursuing the Ph.D. degree with the Department of Computer Science.

His current research interests include logic synthesis, optimization, verification for VLSI designs, and automation for emerging technologies.

Yung-Chih Chen (Member, IEEE) received the B.S., M.S., and Ph.D. degrees from the Department of Computer Science, National Tsing Hua University, Hsinchu, Taiwan, in 2003, 2005, and 2011, respectively.

He is currently an Assistant Professor with the Department of Computer Science and Engineering, Yuan Ze University, Taoyuan, Taiwan. His current research interests include logic synthesis, design verification, and design automation for emerging technologies.

Chun-Yao Wang (Member, IEEE) received the B.S. degree from the Department of Electronics Engineering, National Taipei University of Technology, Taipei, Taiwan, in 1994, and the Ph.D. degree from the Department of Electronics Engineering, National Chiao Tung University, Hsinchu, Taiwan, in 2002.

Since 2003, he has been an Assistant Professor with the Department of Computer Science, National Tsing Hua University, Hsinchu, where he is currently a Distinguished Professor. He has published over 70 technical papers in these areas and is a named inventor in nine patents. His current research interests include logic synthesis, optimization, and verification for very large-scale integrated/system-on-chip designs and emerging technologies.

Dr. Wang's two research results were nominated as Best Papers in the 2009 IEEE Asia and South Pacific Design Automation Conference and the 2010 IEEE/ACM Design Automation Conference. He also was a recipient of the Best Paper Award in 2018 IEEE International Symposium on VLSI Design, Automation and Test.