Verification of Reconfigurable Binary Decision Diagram-Based Single-Electron Transistor Arrays

Yung-Chih Chen, Chun-Yao Wang, *Member, IEEE,* and Ching-Yi Huang

***Abstract*—Recently, single-electron transistors (SETs) have been attracting substantial attention and are considered candidate devices for future integrated circuits due to their ultralow power consumption. To realize SETs, a binary decision diagram-based SET array is proposed as a suitable candidate for implementing Boolean circuits. Then, some works started developing computer-aided design techniques for this new architecture. However, most of them focused on the development of mapping techniques. How to verify the mapping results is still an open problem. Thus, in this paper, we address this problem and develop a satisfiability (SAT)-based verification method. We propose a transformation approach to model the functionality of a mapped SET array as a conjunctive normal form formula. Then, the problem that whether the SET array is functionally equivalent to its specification circuit can be solved with a SAT solver. The experimental results show that the proposed method can successfully verify correct and incorrect SET array implementations with reasonable verification time.**

***Index Terms*—Boolean satisfiability problem, functional equivalence checking, reconfigurable binary decision diagram-based single-electron transistor arrays.**

I. Introduction

SINGLE-ELECTRON transistors (SETs) [8], which work with only a few electrons during their switching operations, have been attracting great attention from researchers in the field of semiconductor nanotechnology due to their ultralow power consumption. Numerous SET demonstrations at the room temperature have also proved that SETs are one of the most possible candidates that could replace conventional complementary metal-oxide-semiconductor (CMOS) devices for future low-power and high-density integrated circuits [10], [14], [15], [18], [20], [21], [23].

Recently, some studies on the design of architectures using SETs were proposed. Because SETs have poor driving capability and poor threshold control due to one or few electron involvement in the switching process, they are not suitable for the conventional CMOS-based logic implementation. To solve this problem, a novel binary decision diagram (BDD)-based [2] logic architecture was proposed as a suitable candidate for implementing Boolean circuits using SETs [1]. A Boolean circuit can be implemented by mapping its BDD onto the BDD-based logic architecture, which is represented as a hexagonal nanowire network controlled by Schottky wrap gates [9], [11].

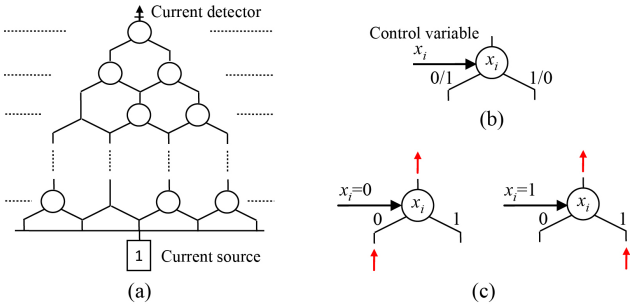

Fig. 1. (a) BDD-based logic architecture using SETs. (b) Node device. (c) Behavior of a node device.

In a hexagonal nanowire network, a node device, which corresponds to a BDD node, has two edges as shown in Fig. 1(a) and (b). It receives the current from the preceding device through either the left or right edges controlled by the variable, and sends the current to the following devices. For example, in Fig. 1(c), when the control variable, x_i, equals 0 (or 1), the node device receives the current through the left (or right) edge. To realize a node device, each edge (left or right) is implemented with a wrap-gate SET device, and a variable is applied to control the conductivity. Additionally, there is a current source, which corresponds to the 1 terminal of a BDD, at the bottom, and there is a current detector at the root. When the input control variables establish a conducting path from the current source to the root so that the current is detected, the output value of the implemented Boolean circuit is 1; otherwise, it is 0. For example, Fig. 2(a) shows an implementation of $a_1 \oplus b_1 \oplus (a_0 b_0)$.

However, the realization of the previous BDD-based logic architecture, as shown in Fig. 2(a), is fixed and not amendable to functional reconfiguration due to the physical etching process involved in its realization. Furthermore, if any of the nanowire segments or the wrap gates is defective, the whole circuit becomes nonfunctional. This is a significant limitation considering that nanowires and few-electron nanodevices traditionally suffered from the variability and reliability issues.

Thus, to increase the flexibility and reliability of the BDD-based logic architecture, a programmable version of SET

Manuscript received September 27, 2012; revised February 6, 2013; accepted May 20, 2013. Date of current version September 16, 2013. This paper was recommended by Associate Editor Y. Chen.

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).

Chun-Yao Wang and Ching-Yi Huang are with the Department of Computer Science, National Tsing Hua University, Hsinchu 30013, Taiwan (e-mail: wcyao@cs.nthu.edu.tw; s9862516@m98.nthu.edu.tw).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCAD.2013.2267453

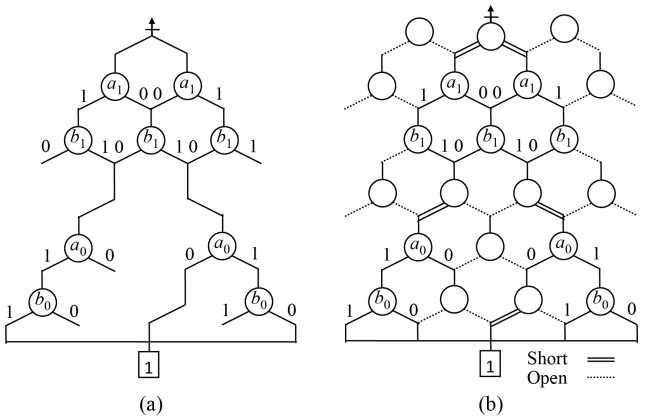

Fig. 2. (a) Example of BDD-based logic architecture using SETs. Missing edges of hexagons are physically etched. (b) Example of reconfigurable BDD-based SET array.

device with wrap-gate-tunable tunnel barriers was proposed [7]. This device can operate in three distinct operation modes: 1) active, 2) open, and 3) short based on the wrap-gate bias voltages, and thus, enables the BDD-based logic architecture to be functionally reconfigurable. For example, Fig. 2(b) shows a reconfigurable BDD-based SET array that also implements $a_1 \oplus b_1 \oplus (a_0 b_0)$. In [17], the simulation to study the electrostatic properties of the programmable SET device was presented and the results showed that it can provide an order of magnitude lower energy-delay than the CMOS device. Furthermore, recent experimental work [12], [13] has presented a wrap-gate SET, and the measured characteristics showed that it is capable of operating in three distinct operation modes. This experimental device can be considered a practical realization of the programmable SET device [7].

In addition to the advances of SET array realization, a computer-aided design (CAD) technique that maps a Boolean circuit onto an SET array was proposed [4] and improved [5], [6]. The technique increases the efficiency of the SET array mapping, which was done with manual efforts [7]. However, no matter what mapping methods (manual or automatic) are used, how to verify the mapping results is still an open problem and an automatic verification method is desirable for the SET arrays to be promising. Thus, in this paper, we address the problem and develop a SAT-based verification method.

Given a mapped SET array A and its specification circuit C, to check their equivalence, we propose a method to transform A into a CNF formula. Then, we transform the verification problem into a Boolean SAT problem, and use a SAT solver, *MiniSat* [24], to determine the equivalence of C and A. However, since a complete CNF formula for a larger SET array could have a great number of variables, making the formula hard to solve, we further introduce a simpler transformation method. The simpler transformation method generates a smaller CNF formula but could increase the number of SAT solving calls required for achieving the verification. Thus, we further exploit the learned counterexamples to speedup the verification process by reducing the number of SAT solving calls.

In the experiments, to show the effectiveness and the efficiency of the proposed method on verifying correct and

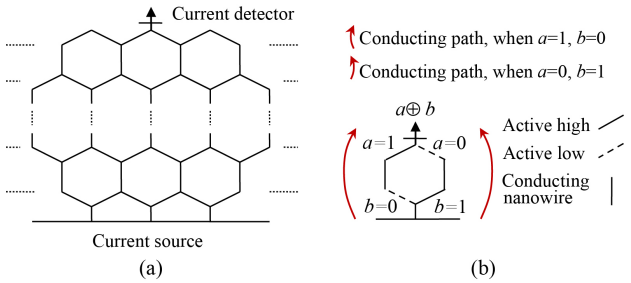

Fig. 3. (a) Reconfigurable BDD-based SET array. (b) Example of $a \oplus b$.

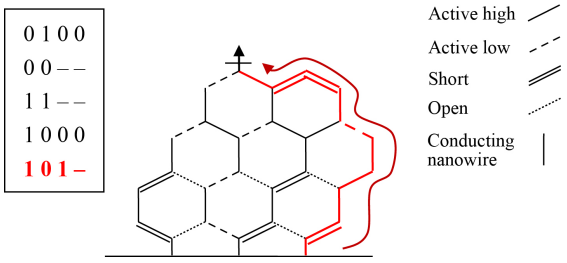

Fig. 4. Example of BDD-based SET array.

incorrect SET array implementations, we used the automatic mapping method [4] to generate correct implementations for a set of MCNC benchmarks [22]. Additionally, we injected errors into the implementations to create incorrect implementations. The experimental results show that the proposed method can successfully and efficiently verify both the correct and the incorrect implementations.

To the best of our knowledge, this is the first work that addresses the verification problem of the reconfigurable BDD-based SET arrays. Due to the unique structure of the SET arrays, SET array verification is more difficult than the conventional combinational verification. The existing verification techniques [16] cannot be directly applied to solve the verification problem. Thus, the main contribution of this paper is making the problem solvable by an existing technique. In addition to the previous synthesis method [4], the proposed verification method is also an important CAD technique for the emerging SET arrays.

The rest of this paper is organized as follows. Section II briefly introduces the reconfigurable BDD-based SET arrays [7], and gives some notations and background. Section III formulates the problem considered in this paper. Section IV presents the proposed verification method with a complete CNF formula. Section V presents the proposed verification method with a simpler CNF formula. Finally, the experimental results and conclusion are presented in Sections VI and VII.

II. Background

A. Reconfigurable BDD-Based SET Array

A reconfigurable BDD-based SET array [7] can be represented as an hexagonal architecture as shown in Fig. 3(a). In the architecture, there is a current detector at the top that measures the current coming from the bottom (current source). All the vertical edges are conducting nanowires. Each sloping

edge corresponds to a programmable SET device and can be configured as active high, active low, short, or open. An active high edge indicates that the corresponding SET device operates in the active mode and is controlled by a variable x. When $x = 1$ (or $x = 0$), the active high edge is conducting (or nonconducting). Conversely, an active low edge is an electrical opposite of an active high edge. The corresponding SET device also operates in the active mode, but is controlled by the complement of x, i.e., x'. A short (or open) edge is electrical short (or open), where the corresponding SET device operates in the short (open) mode. Furthermore, all the active edges (high or low) at the same row are controlled by a same control variable.

A combinational Boolean circuit can be implemented by using an SET array. In the SET array, each control variable corresponds to a primary input (PI) of the Boolean circuit. All the control variables control the conductivities of the active edges, determining whether there exists a conducting path so that the current can pass through, and then be detected by the current detector. For example, Fig. 3(b) shows an SET array implementing $a \oplus b$. When $a \neq b$, the current can be detected by passing through either the left path or the right path. However, if $a = b$, there is no conducting path and the current cannot be detected.

Moreover, Fig. 4 shows another example that implements a Boolean function with the product terms: {(0100), $(00--)$, $(11--)$, (1000), $(101-)$} ($-$ denotes don't care). When the input pattern is $(101-)$, the current can be detected by passing through the highlighted path. According to this example, we can easily observe that an SET array is actually not a BDD, although it is named a BDD-based architecture. For example, a BDD node must have two different edges: one is positive and the other one is negative. However, it is not necessarily true for a node in an SET array. Additionally, an SET array is a planar architecture, which has no crossing edge, but a BDD could have crossing edges. Two adjacent nodes at the same row in an SET array must connect to a same node at the next row, but it is not necessarily true for a BDD.

In this paper, we assume that an SET array has only one root. That is, the specification circuit of the SET array has only one primary output (PO). This assumption is reasonable, because the automatic synthesis method [4] only considers single-output circuits and an n-output circuit is divided into n single-output sub-circuits. Additionally, the proposed verification method can easily be extended to verify an SET array that has multiple roots by considering the roots one by one.

B. Notations

For ease of discussion, we use an abstract graph to represent a reconfigurable BDD-based SET array. Because all the vertical edges in an SET array as shown in Fig. 3(a) are electrical short and nonconfigurable, we only preserve the configurable edges, i.e., the sloping edges, to form a diamond-shaped network as shown in Fig. 5(a). In this diamond-shaped network, the top of a diamond is denoted as a node n. Each node n has a unique location (x, y) with respect to the root node, which represents the current detector and is located at $(0, 0)$. The value of y increases from top to bottom. The value

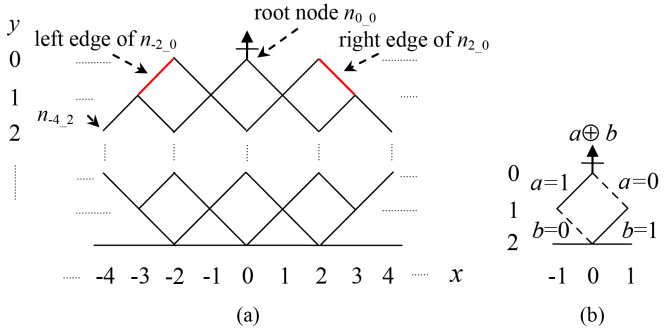

Fig. 5. Abstract diamond-shaped network.

of x increases and decreases from center to right and left, respectively. For convenience, let n_{x_y} denote the node located at (x, y). Furthermore, each node n has a pair of left and right edges. Fig. 5(b) shows the abstract network of the example in Fig. 3(b).

C. Boolean Satisfiability Problem and Tseitin Transformation

In Boolean logic, a formula is said to be in CNF if it is a conjunction of one or more clauses. Each clause is a disjunction of one or more literals, and each literal is a variable or a negated variable. Boolean satisfiability problem is a problem of finding an assignment under which a given CNF formula evaluates to true (i.e., the formula is satisfiable) or proves that there is no such assignment (i.e., the formula is unsatisfiable). Due to the recent advances in SAT solving techniques [24], [25], the Boolean satisfiability problem could be practically tractable, even though it is theoretically intractable.

The Tseitin transformation [19] is a method to transform a Boolean logic function into a CNF formula that represents the logical relationships among the variables in the Boolean logic function. For example, an AND gate with one output variable o and two input variables a and b can be transformed into the following form: $(\neg a \vee \neg b \vee o) \wedge (a \vee \neg o) \wedge (b \vee \neg o)$. For making the formula evaluate to true, the assignments of a, b, and o must satisfy the logical behavior of the AND gate. Similarly, an XOR gate with one output variable o and two input variables a and b can be represented as $(\neg a \vee \neg b \vee \neg o) \wedge (\neg a \vee b \vee o) \wedge (a \vee \neg b \vee o) \wedge (a \vee b \vee \neg o)$.

In this paper, the proposed method for transforming a mapped SET array into a CNF formula works like the Tseitin transformation. With this method, we can transform the verification problem into a Boolean SAT problem.

III. Problem Formulation

The problem formulation of this paper is as follows.

Given a mapped SET array and its specification circuit, verify their functional equivalence. For example, the SET array in Fig. 3(b) can be considered a mapping for $a \oplus b$. Our objective is to verify whether the functionality of the SET array is exactly identical to $a \oplus b$.

To verify the equivalence, a straightforward method is functional simulation. Given an input pattern, the conductivity of each edge is first determined based on the input values.

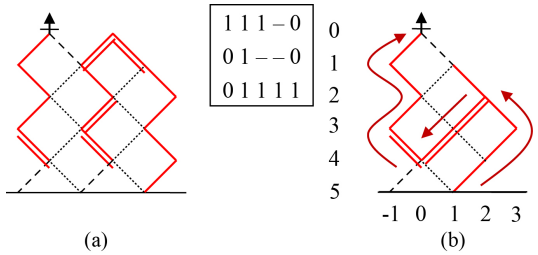

Fig. 6. Example of SET array simulation. (a) Correct implementation. (b) Incorrect implementation.

Then, we can start from the root node to search if there is a conducting path from the root node to the bottom. If so, the output value is one; otherwise, it is 0. For example, Fig. 6(a) and (b) (taken from [4]), respectively, show a correct mapping and an incorrect mapping for a Boolean circuit having the product terms: {(111−0), (01−−0), (01111)}. Let us consider Fig. 6(a) first. Suppose the input pattern is 11111, which is not a minterm. The conducting edges with respect to 11111 are highlighted in bold format. Starting from the root node, we can observe that these conducting edges cannot establish a conducting path reaching the bottom, and thus the output value is 0. However, in Fig. 6(b), for the same input pattern, we can find a conducting path, $n_{0\_0} \rightarrow n_{-1\_1} \rightarrow n_{0\_2} \rightarrow n_{-1\_3} \rightarrow n_{0\_4} \rightarrow n_{1\_3} \rightarrow n_{2\_2} \rightarrow n_{3\_3} \rightarrow n_{2\_4} \rightarrow n_{1\_5}$, from the root node to the bottom. Thus, the current can reach the current detector by passing through the highlighted path and the mapping in Fig. 6(b) is incorrect. With this simulation method, the verification problem can be solved by simulating all the input patterns and checking their output responses one by one. However, one concern about this method is that it could be inefficient for verifying large circuits, since a large number of input patterns need to be simulated.

Formal verification is an alternative and could be a better solution. With the advances of SAT solving techniques [24], [25], efficient SAT-based combinational equivalence checking is becoming increasingly popular in combinational verification. Two circuits under verification are first combined to form a miter [3]. Then, the miter with the output value 1 is transformed into a CNF formula by using the Tseitin transformation. Finally, a SAT solver is used to solve the CNF formula for checking the equivalence of the two circuits. Here, they are nonequivalent if and only if the formula is satisfiable. Inspired by this verification method, if we can transform an SET array into a CNF formula, the verification problem can be easily solved. Thus, we propose a new transformation method for the SET arrays. Due to the unique structure of the SET arrays, the proposed transformation method is much more complicated than the Tseitin transformation.

IV. SET Array Verification

In this section, we first present the method for transforming a mapped SET array into a CNF formula. The complete formula consists of four types of subformulas. We will sequentially introduce them and use some examples to demonstrate their necessity. Then, we present the overall verification flow.

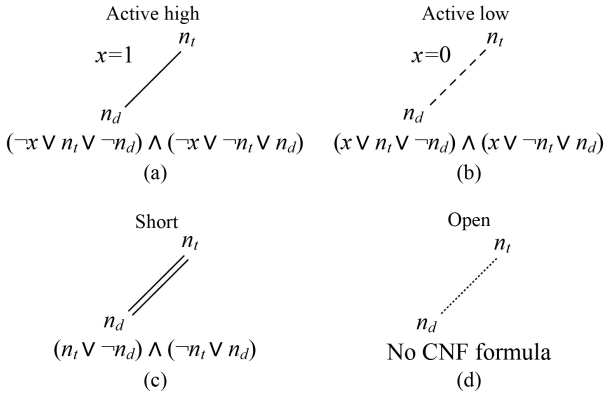

Fig. 7. Type 1 CNF formulas for two adjacent nodes based on four different types of edges. (a) Active high. (b) Active low. (c) Short. (d) Open.

A. Complete CNF Formula

Given a mapped SET array, a complete CNF formula must satisfy the condition that the formula is satisfiable if and only if the input and output values in the reported solution meet the functionality of the SET array. Thus, with a complete CNF formula, we can solve the SET array verification problem with only one SAT solving call. The proposed four types of subformulas as follows are derived based on this condition.

1) *Type 1 CNF Formula:* A mapped SET array is composed of a set of configured edges that determine its functionality. To represent the functionality of the SET array, we can first consider the logical relationship of each pair of adjacent nodes based on the configured edge in between them.

First, let us consider an active high edge as shown in Fig. 7(a). Suppose the active high edge is in between two nodes n_t and n_d, and is controlled by a variable x. According to the behavior of an active high edge mentioned in Section II-A, if $x = 1$, the edge is conducting, and therefore n_t and n_d must have a same logical value. Conversely, if n_t and n_d have different logical values, the edge must be nonconducting and $x = 0$.[1] However, please note that when the edge is nonconducting, i.e., $x = 0$, n_t and n_d do not necessarily have different logical values. Similarly, if n_t and n_d have a same logical value, the edge is not necessarily conducting. For ease of discussion, let $n_t = n_d$ denote n_t and n_d have a same logical value and $n_t \neq n_d$ denote they have different logical values. The logical relationship of x, n_t, and n_d can be represented with the formula: $(\neg x \vee n_t \vee \neg n_d) \wedge (\neg x \vee \neg n_t \vee n_d)$. For making the formula evaluate to true, if $x = 1$, n_t and n_d must have a same logical value. That is, $x = 1$ implies $n_t = n_d$. Conversely, $n_t \neq n_d$ implies $x = 0$.

Next, let us consider an active low edge as shown in Fig. 7(b). Since an active low edge is an electrical opposite of an active high edge, the logical relationship of x, n_t, and n_d can be represented with the formula: $(x \vee n_t \vee \neg n_d) \wedge (x \vee \neg n_t \vee n_d)$. For making the formula evaluate to true, if $x = 0$, n_t and n_d must have a same logical value. Thus, $x = 0$ implies $n_t = n_d$, and $n_t \neq n_d$ implies $x = 1$. Similarly, please note that $x = 1$ does not imply $n_t \neq n_d$.

[1](if P, then Q)≡(if not Q, then not P).

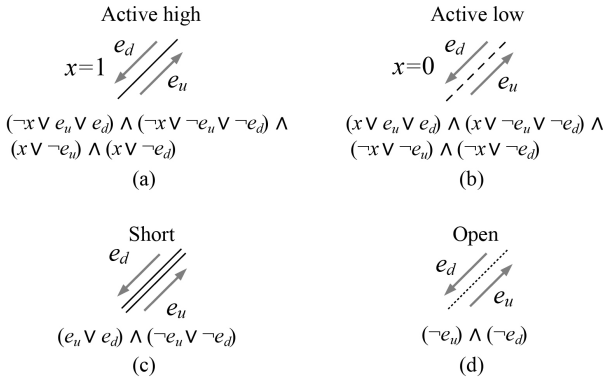

Fig. 8. CNF formulas for four different types of edges. (a) Active high. (b) Active low. (c) Short. (d) Open.

For a short edge as shown in Fig. 7(c), since it is permanently conducting, n_t and n_d always have a same logical value, i.e., $n_t = n_d$. Thus, their logical relationship can be represented with the formula: $(n_t \vee \neg n_d) \wedge (\neg n_t \vee n_d)$. As for an open edge as shown in Fig. 7(d), since n_t and n_d have no direct logical relationship based on the open edge, there is no CNF formula for them.

Finally, for a mapped SET array, its Type 1 CNF formula is the conjunction of the CNF formulas of all the pairs of adjacent nodes in it. Furthermore, since the bottom of an SET array is the current source, the values of the nodes at the bottom are permanently one. The following formula is the Type 1 CNF formula for the SET array in Fig. 5(b), which implements $a \oplus b$:

$$\begin{aligned}(\neg a \vee n_{0\_0} \vee \neg n_{-1\_1}) \wedge (\neg a \vee \neg n_{0\_0} \vee n_{-1\_1}) \wedge \\ (a \vee n_{0\_0} \vee \neg n_{1\_1}) \wedge (a \vee \neg n_{0\_0} \vee n_{1\_1}) \wedge \\ (b \vee n_{-1\_1} \vee \neg n_{0\_2}) \wedge (b \vee \neg n_{-1\_1} \vee n_{0\_2}) \wedge \\ (\neg b \vee n_{1\_1} \vee \neg n_{0\_2}) \wedge (\neg b \vee \neg n_{1\_1} \vee n_{0\_2}) \wedge \\ (n_{0\_2}).\end{aligned} \tag{1}$$

Here, as mentioned in Section II-B, n_{x_y} denotes the node located at (x, y). Because n_{0_2} is the current source, we add the clause (n_{0_2}) to force $n_{0\_2} = 1$ when the formula is satisfied.

According to the example of (1), we can easily observe that Type 1 CNF formula is not a complete CNF formula, because the formula is satisfiable under $a = 1$, $b = 1$, and $n_{0\_0} = 1$. $a = 1$ and $b = 1$ actually generate 0 for $a \oplus b$. One reason for this is that Type 1 CNF formula does not consider the current flow in an SET array. Thus, the situation that n_{0_0} is one but the current actually does not reach n_{0_0} occurs.

2) *Type 2 CNF Formula:* Type 2 CNF formula is derived by considering the current flow on each edge. For each edge e, because when e is conducting, the current flow, if any, can pass through e from either down to up or up to down, we use two variables e_u and e_d to represent the two different directions. Here, $e_u = 1$ means that the current, if any, can pass through e from down to up. Conversely, $e_u = 0$ means that the current cannot pass through e from down to up. Furthermore, $e_d = 1$ means that the current, if any, can pass through e from up to down. Thus, when e is conducting, one, and only one, of e_u and e_d must be one. Conversely, when e is nonconducting, both e_u and e_d must be 0. Thus, for an active high edge controlled by a variable x, the CNF formula for it is $(\neg x \vee e_u \vee e_d) \wedge (\neg x \vee \neg e_u \vee \neg e_d) \wedge (x \vee \neg e_u) \wedge (x \vee \neg e_d)$. For an active low edge, the CNF formula is $(x \vee e_u \vee e_d) \wedge (x \vee \neg e_u \vee \neg e_d) \wedge (\neg x \vee \neg e_u) \wedge (\neg x \vee \neg e_d)$. If e is a short edge, which is permanently conducting, the CNF formula is $(e_u \vee e_d) \wedge (\neg e_u \vee \neg e_d)$. Furthermore, if e is an open edge, which is permanently nonconducting, the CNF formula is $(\neg e_u) \wedge (\neg e_d)$. The CNF formulas for these four different types of edges are summarized in Fig. 8.

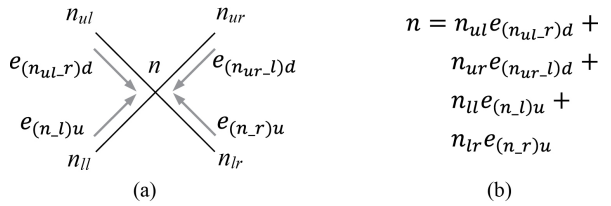

Fig. 9. (a) Node n and its four adjacent nodes, n_{ul}, n_{ur}, n_{ll}, and n_{lr}. (b) Boolean function of n.

Thus, the Type 2 CNF formula is the conjunction of the CNF formulas of all the edges. For example, Fig. 5(b) has four edges (two active high edges and two active low edges), and therefore, the Type 2 CNF formula for this SET array is as follows:

$$\begin{aligned}(\neg a \vee e_{(n_{0\_0}\_l)u} \vee e_{(n_{0\_0}\_l)d}) \wedge (\neg a \vee \neg e_{(n_{0\_0}\_l)u} \vee \neg e_{(n_{0\_0}\_l)d}) \wedge \\ (a \vee \neg e_{(n_{0\_0}\_l)u}) \wedge (a \vee \neg e_{(n_{0\_0}\_l)d}) \\ (a \vee e_{(n_{0\_0}\_r)u} \vee e_{(n_{0\_0}\_r)d}) \wedge (a \vee \neg e_{(n_{0\_0}\_r)u} \vee \neg e_{(n_{0\_0}\_r)d}) \wedge \\ (\neg a \vee \neg e_{(n_{0\_0}\_r)u}) \wedge (\neg a \vee \neg e_{(n_{0\_0}\_r)d}) \\ (b \vee e_{(n_{-1\_1}\_r)u} \vee e_{(n_{-1\_1}\_r)d}) \wedge (b \vee \neg e_{(n_{-1\_1}\_r)u} \vee \neg e_{(n_{-1\_1}\_r)d}) \wedge \\ (\neg b \vee \neg e_{(n_{-1\_1}\_r)u}) \wedge (\neg b \vee \neg e_{(n_{-1\_1}\_r)d}) \\ (\neg b \vee e_{(n_{1\_1}\_l)u} \vee e_{(n_{1\_1}\_l)d}) \wedge (\neg b \vee \neg e_{(n_{1\_1}\_l)u} \vee \neg e_{(n_{1\_1}\_l)d}) \wedge \\ (b \vee \neg e_{(n_{1\_1}\_l)u}) \wedge (b \vee \neg e_{(n_{1\_1}\_l)d}).\end{aligned} \tag{2}$$

Here, $e_{(n_{x\_y}\_l)}$ and $e_{(n_{x\_y}\_r)}$ denote the left and the right edges of the node n_{x_y}, respectively.

3) *Type 3 CNF Formula:* Next, we consider the condition for a node n to be one, i.e., n receives the current. Without loss of generality, we assume that n has four adjacent nodes: the upper-left node n_{ul}, the upper-right node n_{ur}, the lower-left node n_{ll}, and the lower-right node n_{lr} as shown in Fig. 9(a). The variables representing the current flow on the corresponding edges are shown as well. For n to be one, the current must pass through at least one out of n_{ul}, n_{ur}, n_{ll}, and n_{lr}, and then reaches n. For example, if the current passes through n_{ul} and n_{ul}'s right edge $e_{(n_{ul}\_r)}$, i.e., $n_{ul} = e_{(n_{ul}\_r)d} = 1$, then n is one. Similarly, if $n_{ur} = e_{(n_{ur}\_l)d} = 1$, $n_{ll} = e_{(n\_l)u} = 1$, or $n_{lr} = e_{(n\_r)u} = 1$, n is one as well. However, if the current cannot reach n, n is 0.

Thus, we can use a Boolean function to represent the functionality of n based on its adjacent nodes: $n = n_{ul}e_{(n_{ul}\_r)d} + n_{ur}e_{(n_{ur}\_l)d} + n_{ll}e_{(n\_l)u} + n_{lr}e_{(n\_r)u}$ as shown in Fig. 9(b). Then, the CNF formula of n can be obtained by using the Tseitin transformation to transform the Boolean function into a CNF formula. Finally, the Type 3 CNF formula is the conjunction of the CNF formulas of all the nodes. Here, we use $Tseitin(n_i)$ to

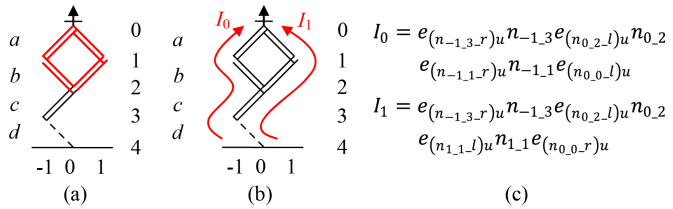

Fig. 10. Cyclic conducting path problem. (a) Example of cyclic conducting path. (b) Two paths I_0 and I_1. (c) Boolean functions of I_0 and I_1.

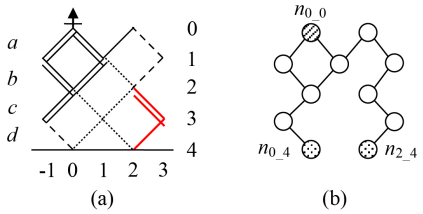

Fig. 11. Example of finding all possible paths between root and bottom. (a) Mapped SET array. (b) Undirected graph with respect to array in (a).

denote the CNF formula of a node n_i and use $\bigwedge_{n_i \in A} Tseitin(n_i)$ to denote the Type 3 CNF formula.

Before introducing Type 4 formula, let us consider the example in Fig. 5(b) again. If we use the CNF formulas of the Types 1–3 to represent the SET array, the formula is Formula (1) $\wedge$ Formula (2) $\wedge \bigwedge_{n_i \in A} Tseitin(n_i)$, and the formula can correctly represent the SET array. For example, when $a \neq b$, the formula is satisfiable only under $n_{0\_0} = 1$, which can be easily verified based on (1). Furthermore, when $a = 1$ and $b = 1$, the formula is satisfiable only under $n_{0\_0} = 0$. This is because $n_{-1\_1} = 1$ and $e_{(n_{0\_0}\_l)u} = 1$ are necessary for n_{0_0} to be one, when $a = 1$ and $b = 1$. However, since $e_{(n_{-1\_1}\_r)}$ is nonconducting, n_{-1_1} is 0, and thus n_{0_0} is 0. Similarly, when $a = 0$ and $b = 0$, the formula is satisfiable only under $n_{0\_0} = 0$.

Although the formula is complete for the SET array in Fig. 5(b), there are some other cases that the formula cannot work for. Let us consider the SET array in Fig. 10(a). When the input pattern is ($a = 1$, $b = 0$, $c = 1$, $d = 1$), the output value is 0. However, if we use the CNF formulas of the Types 1−3 to represent the SET array, the formula is satisfiable under $n_{0\_0} = 1$, when $a = 1$, $b = 0$, $c = 1$, and $d = 1$. This is because the four conducting edges highlighted in bold format, $e_{(n_{0\_0}\_r)}$, $e_{(n_{0\_0}\_l)}$, $e_{(n_{-1\_1}\_r)}$, and $e_{(n_{1\_1}\_l)}$, form a cyclic conducting path. Here, $n_{-1\_1} = 1$ and $e_{(n_{0\_0}\_l)u} = 1$ result in $n_{0\_0} = 1$. $n_{0\_0} = 1$ and $e_{(n_{0\_0}\_r)d} = 1$ result in $n_{1\_1} = 1$. $n_{1\_1} = 1$ and $e_{(n_{1\_1}\_l)d} = 1$ result in $n_{0\_2} = 1$. $n_{0\_2} = 1$ and $e_{(n_{-1\_1}\_r)u} = 1$ result in $n_{-1\_1} = 1$. There is no conflict among the sufficient conditions for n_{0_0}, n_{-1_1}, n_{0_2}, and n_{1_1} to be one. Thus, they can be one without receiving the current.

To solve this cyclic conducting path problem, we need to consider whether there really exists a conducting path from the bottom to the root. If so, n_{0_0} is one; otherwise, n_{0_0} is 0. That is, we need a CNF formula that considers all the possible paths from the bottom to the root. Type 4 CNF formula is derived based on this requirement.

4) *Type 4 CNF Formula:* For ease of discussion, we first use the example in Fig. 10(a) to demonstrate Type 4 CNF formula. In this SET array, there are totally two possible paths from the bottom to the root. Let I_0 and I_1 denote these two paths as shown in Fig. 10(b). When I_0 is one and I_1 is one, I_0 and I_1 are conducting, respectively. Thus, the functionality of I_0 can be represented as $I_0 = e_{(n_{-1\_3}\_r)u} n_{-1\_3} e_{(n_{0\_2}\_l)u} n_{0\_2} e_{(n_{-1\_1}\_r)u} n_{-1\_1} e_{(n_{0\_0}\_l)u}$ as shown in Fig. 10(c). Additionally, the functionality of I_1 can be represented as $I_1 = e_{(n_{-1\_3}\_r)u} n_{-1\_3} e_{(n_{0\_2}\_l)u} n_{0\_2} e_{(n_{1\_1}\_l)u} n_{1\_1} e_{(n_{0\_0}\_r)u}$.

Furthermore, when n_{0_0} is one, at least one of I_0 and I_1 must be one. Thus, the Type 4 CNF formula for the SET array is $(\neg n_{0\_0} \vee I_0 \vee I_1) \wedge Tseitin(I_0) \wedge Tseitin(I_1)$. Here, $Tseitin(I_i)$ denotes the CNF formula of I_i obtained by using the Tseitin transformation.

To compute the Type 4 CNF formula for an SET array, we need to identify all the possible paths from the bottom to the root in the array. A computation method is as follows. First, an SET array is seen as an undirected graph. Because an open edge is permanently nonconducting, the edges with respect to the open edges can be removed. Then, we can find all the possible paths from the root to one of the nodes at the bottom by using the depth-first search (DFS) starting from the root. For example, the SET array in Fig. 11(a) can be modeled as an undirected graph as shown in Fig. 11(b). Here, the open edges are removed. Starting from n_{0_0}, we use the DFS to find all the possible paths that can reach either n_{0_4} or n_{2_4}. In this example, there are totally four possible paths.

Finally, the complete CNF formula for a mapped SET array is the conjunction of the CNF formulas of the Types 1−4. For example, the complete CNF formula for the SET array in Fig. 5(b) is Formula (1) $\wedge$ Formula (2) $\wedge \bigwedge_{n_i \in A} Tseitin(n_i) \wedge (\neg n_{0\_0} \vee I_0 \vee I_1) \wedge Tseitin(I_0) \wedge Tseitin(I_1)$. Here, I_0 and I_1 denote the two conducting paths (left and right) in the SET array.

Let us consider the correctness of the complete CNF formula. For an input pattern that results in a conducting path from n_{0_0} to the bottom, $n_{0\_0} = 1$ is necessary for making the formula evaluate to true due to the Type 1 CNF formula. Furthermore, for an input pattern that does not result in any conducting path from n_{0_0} to the bottom, i.e., all the possible paths are nonconducting, $n_{0\_0} = 0$ is necessary for making the formula evaluate to true due to the Type 4 CNF formula. Thus, the functionality of a mapped SET array can be correctly represented by the complete CNF formula.

Because the conventional combinational circuits are acyclic, the Tseitin transformation can work without considering the cyclic path problem. However, the cyclic conducting path problem could happen in an SET array. Thus, the proposed transformation method is more complicated than the Tseitin transformation. It also implies that the verification of an SET array is more difficult than that of a conventional combinational circuit.

B. Overall Verification Flow

With the proposed transformation method, we can solve the SET array verification problem with the verification flow

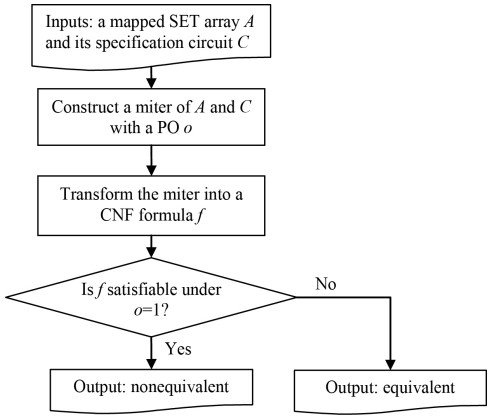

Fig. 12. Verification flow.

as shown in Fig. 12. Given a mapped SET array and its specification circuit, we first construct a miter by connecting their POs with an additional XOR gate o. Next, we transform the miter into a CNF formula with the proposed transformation method and the Tseitin transformation. Then, we use a SAT solver to check whether the formula is satisfiable or not under the constraint that o equals one. Finally, if the formula is satisfiable, the SET array implementation is incorrect; otherwise, it is correct.

Although the above method can solve the verification problem, it could be practically inefficient for large SET arrays. This is because constructing a complete CNF formula involves identifying all the possible paths from the root to the bottom, which is a computation-intensive process. Additionally, modeling all these possible paths could create a very large CNF formula. Thus, we further propose a simpler formula to replace the Type 4 CNF formula for reducing the computational complexity. This simpler formula results in a smaller formula but increases the number of SAT solving calls required for completing the verification process. To reduce the number of SAT solving calls, we further propose a method to make the best use of the counterexamples.

V. Enhanced Verification Flow

In this section, we first present the simpler CNF formula. Then, we introduce how to use the counterexamples to reduce the number of SAT solving calls. Finally, we present the verification flow based on the simpler CNF formula.

A. Simpler CNF Formula

Unlike the Type 4 CNF formula, a simpler formula models the partial paths between two rows rather than the full paths from the root to the bottom. For example, suppose the two rows in Fig. 13(a) are two adjacent rows in an SET array. Here, there are four edges between these two rows, and thus, there are totally four paths that the current from the lower row can pass through to reach the upper row as shown in Fig. 13(b). Because the current from the bottom must pass each row in an SET array for reaching the root, at least one of the four partial paths must be conducting for n_{0_0} to be one. Let

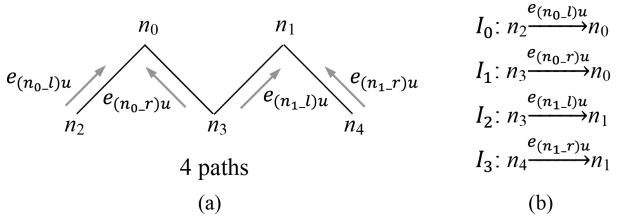

Fig. 13. Example of all partial paths between two adjacent rows. (a) Four partial paths between rows. (b) Details of partial paths.

I_0, I_1, I_2, and I_3 denote these four partial paths, respectively. The functionalities of I_0, I_1, I_2, and I_3 can be represented as $I_0 = n_2 e_{(n_0\_l)u} n_0$, $I_1 = n_3 e_{(n_0\_r)u} n_0$, $I_2 = n_3 e_{(n_1\_l)u} n_1$, and $I_3 = n_4 e_{(n_1\_r)u} n_1$. Since at least one of $I_0 = 1$, $I_1 = 1$, $I_2 = 1$, and $I_3 = 1$ is necessary for n_{0_0} to be one, the CNF formula for these two adjacent rows is $(\neg n_{0\_0} \vee I_0 \vee I_1 \vee I_2 \vee I_3) \wedge Tseitin(I_0) \wedge Tseitin(I_1) \wedge Tseitin(I_2) \wedge Tseitin(I_3)$.

For a mapped SET array, the CNF formula is the conjunction of the CNF formulas for each two adjacent rows in the array. This CNF formula can be used to replace the Type 4 CNF formula to form a simpler CNF formula in the proposed verification flow. However, because not all the possible paths from the bottom to the root are modeled in the simpler formula, there may exist some invalid solutions that satisfy the simpler formula. That is, there may exist an input pattern that actually generates 0 for the SET array but satisfies the simpler formula under $n_{0\_0} = 1$. Thus, in the new verification flow, when the CNF formula for the miter is satisfied, we need to check whether the reported solution is valid or not. Please note that for an input pattern that results in a conducting path from n_{0_0} to the bottom, $n_{0\_0} = 1$ is still necessary for making the simpler formula evaluate to true due to the Type 1 CNF formula. That is, there is no input pattern that actually generates one for the SET array but satisfies the simpler formula under $n_{0\_0} = 0$.

The new verification flow is shown in Fig. 14. Compared to the original flow in Fig. 12, the main difference is that when the formula is satisfied, we check whether the reported solution s is valid or not. If s is valid, the SET array implementation is incorrect. Conversely, If s is invalid, i.e., s is a counterexample, we add the corresponding clause into the formula to prevent the SAT solver from finding s again, and then repeat the SAT solving process.

Although the size of the formula decreases in the new verification flow, the flow possibly spends much time on iteratively finding invalid solutions, which affects its efficiency. To reduce the number of invalid solutions, we can increase the row count k of the adjacent rows under consideration and model all the partial paths between the lowest and the highest rows in the k adjacent rows; that is, to make the CNF formula more elaborate. Similarly, at least one conducting path from the lowest row to the highest row in the k adjacent rows is necessary for n_{0_0} to be 1.

Fig. 15(a) shows an example of considering three adjacent rows. Suppose these three rows have three, two, and three nodes, respectively. There are totally eight partial paths between the lowest and the highest rows. Furthermore, for

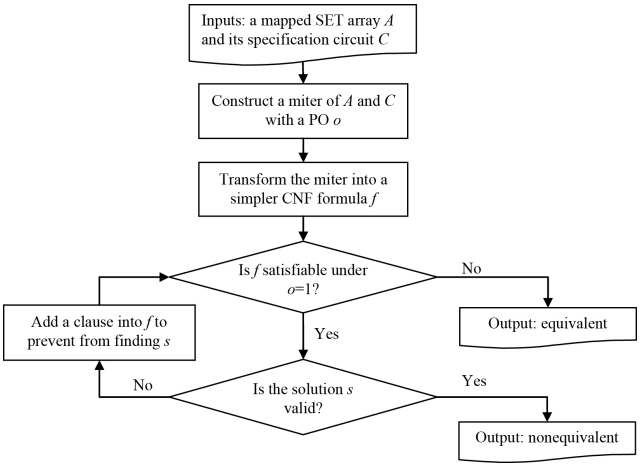

Fig. 14. Verification flow with a simpler CNF formula.

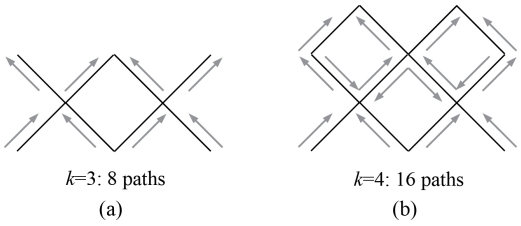

Fig. 15. Examples of all partial paths between lowest and highest rows in (a) three adjacent rows and (b) four adjacent rows.

the example of considering four adjacent rows in Fig. 15(b), there are totally 16 paths from the lowest row to the highest row. These paths can be identified by computing all the paths between each pair of nodes: one is at the highest row and the other one is at the lowest row. Here, because a conducting edge is bidirectional, before the current reaches the highest row, it can pass through the edges in the middle row from up to down. Thus, both the two directions for the edges in the middle row need to be considered. However, for the edges in the highest and the lowest rows, modeling two directions for them is not necessary. This is because in our modeling, exactly one node in the highest and the lowest rows of the considered k adjacent rows can be reached by the current flow.

In general, increasing the row count k could reduce the number of counterexamples but increases the size of the CNF formula due to more partial paths. Thus, there is a tradeoff between the number of counterexamples and the size of the CNF formula. When k equals the height of the SET array, the resultant CNF formula is identical to the Type 4 CNF formula if we model all the paths from the bottom to the root.

In addition to increasing k, another method is to use the learned counterexample to prune the other counterexamples from the solution space. This method to be presented in the next subsection could reduce the number of SAT solving calls without largely increasing the size of the CNF formula.

B. Counterexample Addition

Suppose an SET array has l control variables, $x_0 \sim x_{l-1}$, and $(x_0 = v_0, x_1 = v_1, ..., x_{l-1} = v_{l-1})$, where $v_0 \sim v_{l-1}$ are either 0 or one, is a counterexample reported by the SAT solver. To prevent the SAT solver from finding the same solution again, we generally add the clause $(s_0 \vee s_1 \vee ... s_{l-1})$ into the CNF formula. Here, s_i is $\neg x_i$ if $v_i = 1$; otherwise, s_i is x_i.

For example, let us consider the SET array in Fig. 11(a). The input pattern (a = 1, b = 0, c = 1, d = 1) can make a simpler CNF formula with k = 2 evaluate to true under $n_{0\_0} = 1$, but it actually does not result in a conducting path from the bottom to the root. Suppose we have identified that it is an invalid solution, we add the clause $(\neg a \vee b \vee \neg c \vee \neg d)$ into the CNF formula to force the SAT solver to find the other solutions. To determine whether the input pattern (a = 1, b = 0, c = 1, d = 1) is invalid or not, we can check if $(a) \wedge (\neg b) \wedge (c) \wedge (d) \wedge (\neg n_{0\_0}) \wedge f_{T1}$ is satisfiable. If so, it is an invalid solution. Here, f_{T1} denotes the Type 1 CNF formula of the SET array.

With this method, only the identified counterexample can be pruned at each iteration. To make the better use of the counterexample, we can add the clauses by considering the conductivity of each edge, so that more than one solution could be pruned at each iteration.

Let us consider the example in Fig. 11(a) again. We know that the input pattern (a = 1, b = 0, c = 1, d = 1) is an invalid solution. For this input pattern, $e_{(n_{-1\_3\_}r)}$, $e_{(n_{2\_0\_}r)}$, and all the open edges are nonconducting in the SET array. Because (a = 1, b = 0, c = 1, d = 1) is invalid, all the input patterns that simultaneously cause $e_{(n_{-1\_3\_}r)}$ and $e_{(n_{2\_0\_}r)}$ to be nonconducting cannot result in a conducting path either. Thus, these input patterns must be invalid as well if they make the simpler CNF formula evaluate to true under n_{0_0} = 1. In other words, a valid input pattern that results in a conducting path must cause at least one of $e_{(n_{-1\_3\_}r)}$ and $e_{(n_{2\_0\_}r)}$ to be conducting. Thus, we can create a clause to force the SAT solver to find a solution that satisfies the requirement.

To determine whether an edge e is nonconducting under an invalid input pattern, we can check if both e_u and e_d are 0 in the reported solution. If so, e is nonconducting; otherwise e is conducting. Additionally, to make an edge e conducting, an input pattern must result in either e_u = 1 or e_d = 1. Thus, suppose an invalid pattern results in l nonconducting edges, $e_{(0)} \sim e_{(l-1)}$, except the open edges, we add the clause $(e_{(0)u} \vee e_{(0)d} \vee ... e_{(l-1)u} \vee e_{(l-1)d} \vee \neg n_{0\_0})$ into the CNF formula to force the SAT solver to find an input pattern that can make at least one out of the l nonconducting edges conducting for n_{0_0} to be one.

In the example in Fig. 11(a), after identifying that the input pattern (a = 1, b = 0, c = 1, d = 1) is invalid, we add $(e_{(n_{2\_0\_}r)u} \vee e_{(n_{2\_0\_}r)d} \vee e_{(n_{-1\_3\_}r)u} \vee e_{(n_{-1\_3\_}r)d} \vee \neg n_{0\_0})$ rather than $(\neg a \vee b \vee \neg c \vee \neg d)$ into the CNF formula. Then, in addition to the input pattern itself, the solutions with (a = 1, b = 0, c = 0, d = 1), (a = 1, b = 1, c = 0, d = 1), and (a = 1, b = 1, c = 1, d = 1) can be pruned as well. This is because they make $e_{(n_{-1\_3\_}r)}$ and $e_{(n_{2\_0\_}r)}$ nonconducting simultaneously.

VI. Experimental Results

We implemented the proposed verification method in C language. The experiments were conducted on a 3.0 GHz

TABLE I
CPU TIME FOR VERIFYING CORRECT SET ARRAY IMPLEMENTATIONS

Benchmark	$\|PI\|$	$\|PO\|$	$\|Hex.\|$	$M_{k=0}$	$M_{k=2}$	$M_{k=3}$	$M_{k=4}$
C17	5	2	16	0.01	0.01	0.01	0.01
cm138a	6	8	167	0.01	0.01	0.01	0.01
x2	10	7	109	0.01	0.01	0.01	0.01
cm85a	11	3	290	0.01	0.01	0.01	0.01
cm151a	12	2	110	0.01	0.01	0.01	0.01
cm162a	14	5	584	0.01	0.01	0.01	0.01
cu	14	11	259	0.01	0.01	0.01	0.02
cmb	16	4	228	0.01	0.01	0.01	0.01
cm163a	16	5	489	0.01	0.01	0.01	0.02
pm1	16	13	475	0.01	0.01	0.01	0.01
pcle	19	9	254	0.01	0.01	0.02	0.01
cc	21	20	617	0.01	0.01	0.01	0.01
i1	25	16	635	0.01	0.01	0.01	0.01
pcler8	27	17	1038	0.04	0.04	0.02	0.03
c8	28	18	894	0.01	0.01	0.02	0.02
unreg	36	16	1288	0.02	0.01	0.02	0.02
term1	34	10	34246	5.57	8.21	6.47	9.86
count	35	16	1936	0.04	0.02	0.04	0.05
b9	41	21	4383	0.07	0.07	0.05	0.12
cht	47	36	2380	0.03	0.03	0.03	0.04
apex7	49	37	12435	0.36	0.44	0.44	0.71
example2	85	66	13425	0.19	0.23	0.21	0.29
Total				6.46	9.19	7.44	11.29

Linux platform (CentOS 4.8). The experiments include two parts. The first one focuses on verifying correct SET array implementations and the second one focuses on verifying incorrect SET array implementations. To generate correct implementations, we used the automatic mapping approach [4] to map a set of benchmarks from the MCNC benchmark suite [22]. Since the SET array under consideration has only one PO, we mapped the function associated with each PO in a benchmark once at a time. Additionally, for each correct implementation, we injected different errors into it to generate incorrect implementations.

To show the importance of the Type 4 CNF formula and its effect on the verification efficiency, we implemented the proposed method with four versions. In the first version, the CNF formula for an SET array consists of the formulas of the Types 1–3 only without the Type 4 CNF formula, and this version is denoted as $M_{k=0}$. In the second version denoted as $M_{k=2}$, the CNF formula involves the Type 4 CNF formula with $k = 2$. That is, the partial paths between two adjacent rows are considered in this version. In the third and the fourth versions denoted as $M_{k=3}$ and $M_{k=4}$, the CNF formulas involve the Type 4 CNF formula with $k = 3$ and $k = 4$, respectively.

Table I summarizes the experimental results of verifying the correct SET array implementations. Column 1 lists the benchmarks. Columns 2 and 3 list the numbers of PIs and POs in each benchmark, respectively. Column 4 lists the size of each SET array implementation in terms of the number of hexagons. Columns 5–8 list the spent verification time measured in second by the four versions of the proposed verification method, respectively.

The experimental results show that all the four versions can efficiently verify the correct implementations. Only the benchmark *term*1 costs more than one second. This efficiency comes from the fact that the automatic mapping approach does not result in cyclic conducting paths in a correct SET array implementation, and thus all the four versions require only one SAT solving call for solving the verification problem. In general, when the value of k increases, the verification time increases as well. This is because a larger k results in a larger formula. Thus, $M_{k=0}$, which always constructs a smallest formula among the four versions, is better than the others for verifying the correct implementations. However, it is possible that a larger formula is easier to be solved than a smaller formula when the larger formula has a smaller search space. It is the reason why $M_{k=3}$ is better than $M_{k=2}$ in the experiments.

Next, let us consider the experiments of verifying incorrect SET array implementations. To inject an error into an SET array implementation, we randomly select one node and one of its edges, and then change the status of the edge. For example, if the status of a randomly selected edge is active high, active low, or open, we change it to short. However, if the status is short, we change it to active high. For comparison, we generate four different types of incorrect implementations by injecting different numbers of errors: one, three, five, and ten errors. Similarly, we use the four different versions, $M_{k=0}$, $M_{k=2}$, $M_{k=3}$, and $M_{k=4}$, to verify the incorrect implementations. Each benchmark is repeatedly executed ten times for measuring the average results.

The experimental results show that all the four versions can successfully identify the incorrect implementations. Additionally, they are still very efficient for most of the benchmarks in the same benchmark set. Thus, we only show and discuss the experimental results of the benchmarks that cost more than one second for verification to be completed.

Table II shows the experimental results of the benchmark *term*1. Column 1 lists the numbers of injected errors. Columns 2, 4, 6, and 8 list the spent verification time measured in second by $M_{k=0}$, $M_{k=2}$, $M_{k=3}$, and $M_{k=4}$, respectively. Furthermore, the numbers of solving calls required by the four versions of the method are listed in Columns 3, 5, 7, and 9, respectively.

According to Table II, we can observe that $k = 2$ is the best choice for verifying most of the incorrect implementations. Please note that finding out a valid solution is enough to determine the implementation is incorrect. Thus, the verification time depends on how much time is spent to find out a valid solution. When the number of errors is fixed, although increasing the value of k can reduce the number of invalid solutions, it does not necessarily imply that a valid solution can be identified more quickly. Thus, it is possible that $M_{k=4}$ requires more SAT solving calls than $M_{k=3}$ for some cases, and therefore, spends more verification time, as can be seen in the five-error case of this benchmark.

Furthermore, when the value of k is fixed, increasing the number of errors could result in more cyclic conducting paths, increasing the number of invalid solutions, and thus more verification time is required. However, increasing the number of errors could also result in more valid solutions, increasing the possibility that the SAT solver finds a valid solution. Thus, some cases show that the verification time decreases when the number of errors increases. As a result, there is no direct relationship between the number of errors and the verification time.

TABLE II
EXPERIMENTAL RESULTS OF VERIFYING INCORRECT SET ARRAY IMPLEMENTATIONS FOR BENCHMARK *term*1

$\|E\|$	$M_{k=0}$		$M_{k=2}$		$M_{k=3}$		$M_{k=4}$	
	T (s)	$\|SAT\|$	T (s)	$\|SAT\|$	T (s)	$\|SAT\|$	T (s)	$\|SAT\|$
1	283.2	20161.0	193.4	6250.5	262.1	7352.8	234.2	6148.8
3	837.7	35110.5	406.8	13332.7	531.9	15533.5	442.4	12093.3
5	868.5	27911.0	754.5	22042.0	801.0	20930.8	807.5	21559.5
10	89.2	4139.5	106.4	3798.5	205.6	6557.2	182.3	5428.3

TABLE III
EXPERIMENTAL RESULTS OF VERIFYING INCORRECT SET ARRAY IMPLEMENTATIONS FOR BENCHMARK *count*

$\|E\|$	$M_{k=0}$		$M_{k=2}$		$M_{k=3}$		$M_{k=4}$	
	T (s)	$\|SAT\|$	T (s)	$\|SAT\|$	T (s)	$\|SAT\|$	T (s)	$\|SAT\|$
1	591.7	110857.3	517.9	85149.3	562.8	85103.3	521.9	85092.8
3	0.1	37.5	0.1	115.3	0.1	126.8	0.6	1821.8
5	0.1	19.5	0.1	18.2	0.1	19.8	0.1	17.8
10	0.1	17.7	0.1	17.2	0.1	17.5	0.1	16.2

TABLE IV
EXPERIMENTAL RESULTS OF VERIFYING INCORRECT SET ARRAY IMPLEMENTATIONS FOR BENCHMARK *apex*7

$\|E\|$	$M_{k=0}$		$M_{k=2}$		$M_{k=3}$		$M_{k=4}$	
	T (s)	$\|SAT\|$	T (s)	$\|SAT\|$	T (s)	$\|SAT\|$	T (s)	$\|SAT\|$
1	1576.2	99746.4	847.4	49655.4	347.1	26201.4	463.9	26332.2
3	>3262.1	>174816.4	730.9	53131.2	742.3	52619.4	883.5	53614.2
5	313.9	22695.0	62.9	5695.0	217.4	16037.6	1000.3	55715.0
10	1.3	592.2	2.1	473.4	2.6	346.2	2.5	271.8

Tables III and IV show the results of the benchmarks *count* and *apex*7, respectively. In Table III, $k = 2$ is the best choice as well. Furthermore, when the number of errors is larger than or equals three, the nonequivalence is very easy to be detected by all the four versions. Thus, only a little verification time is spent. In Table IV, there exist some cases for which $M_{k=0}$ cannot complete their verification within the time limit, 5000 seconds. Thus, the Type 4 CNF formula is necessary for efficiently verifying these cases. Similarly, when the number of errors is ten, only a little time is required for completing the verification by all the four versions.

In summary, the proposed method can successfully verify both the correct and incorrect SET array implementations. Additionally, we practically do not need to construct a complete CNF formula for an SET array. A simpler CNF formula with a small value of k, two or three, is good enough for solving the verification problem.

VII. CONCLUSION

In this paper, we proposed a SAT-based verification method for the reconfigurable SET arrays. The proposed method solved a problem that, in the past, designers did not know whether a complicated SET array implementation is functionally correct or not. We proposed a transformation method for translating an SET array into a CNF formula. Then, the equivalence checking problem of an SET array implementation and its specification circuit was transformed into a Boolean SAT problem, which was solved with a SAT solver. Since a complete CNF formula for an SET array could be very large and hard to construct, we further introduced a simpler CNF formula. Moreover, because using the simpler CNF formula could result in counterexamples, an effective counterexample-addition method was engaged to enhance the performance by pruning the search space. The experimental results showed that the proposed method can successfully and efficiently verify both correct and incorrect SET array implementations.

As the SET technology advances, the development of related CAD tools is becoming more and more important. This paper addressed one of the key issues, design verification, in the SET array-based design flow, and, therefore, the proposed method is a key enabler for using the promising SET arrays as the design platform.

REFERENCES

[1] N. Asahi, M. Akazawa, and Y. Amemiya, "Single-electron logic device based on the binary dicision diagram," *IEEE Trans. Electron. Devices*, vol. 44, no. 7, pp. 1109–1116. Jul. 1997.
[2] R. Bryant, "Graph-based algorithms for Boolean function manipulation," *IEEE Trans. Comput.*, vol. 35, no. 8, pp. 677–691, Aug. 1986.
[3] D. Brand, "Verification of large synthesized designs," in *Proc. Int. Conf. Computer-Aided Design*, 1993, pp. 534–537.
[4] Y. C. Chen, S. Eachempati, C. Y. Wang, S. Datta, Y. Xie, and V. Narayanan, "Automated mapping for reconfigurable single-electron transistor arrays," in *Proc. Design Autom. Conf.*, 2011, pp. 878–883.
[5] Y. C. Chen, S. Eachempati, C. Y. Wang, S. Datta, Y. Xie, and V. Narayanan, "A synthesis algorithm for reconfigurable single-electron transistor arrays," *ACM J. Emerging Technol. Comput. Syst.*, vol. 9, pp. 5:1–5:20, Feb. 2013.
[6] C. E. Chiang, L. F. Tang, C. Y. Wang, C. Y. Huang, Y. C. Chen, S. Datta, and V. Narayanan, "On reconfigurable single-electron transistor arrays synthesis using reordering techniques," in *Proc. Design, Autom. Test Eur. Conf.*, 2013, pp. 1807–1812.

[7] S. Eachempati, V. Saripalli, V. Narayanan, and S. Datta, "Reconfigurable BDD-based quantum circuits," in *Proc. Int. Symp. Nanoscale Arch.*, 2008, pp. 61–67.
[8] M. H. Devoret and H. Grabert, "Introduction to single charge tunneling," in *Single Charge Tunneling*, H. Grabert and M. H. Devoret, Eds. New York, NY, USA: Plenum, 1992, pp. 1–19.
[9] H. Hasegawa and S. Kasai, "Hexagonal binary decision diagram quantum logic circuits using Schottky in-plane and wrap gate control of GaAs and InGaAs nanowires," *Physica E: Low-dimensional Syst. Nanostructures*, vol. 11, pp. 149–154, Oct. 2001.
[10] P. S. Karre, P. L. Bergstrom, G. Mallick, and S. P. Karna, "Room temperature operational single-electron transistor fabricated by focused ion beam deposition," *Appl. Phys. Lett.*, vol. 102, no. 2, pp. 024316-1–024316-3, 2007.
[11] S. Kasai, M. Yumoto, and H. Hasegawa, "Fabrication of GaAs-based integrated 2-bit half and full adders by novel hexagonal BDD quantum circuit approach," in *Proc. Int. Symp. Semiconductor Device Res.*, 2001, pp. 622–625.
[12] L. Liu, V. Saripalli, E. Hwang, V. Narayanan, and S. Datta, "Multigate modulation doped In0.7Ga0.3As quantum well FET for ultra low power digital logic," *Electro Chem. Soc. Trans.*, vol. 35, no. 3, pp. 311–317, 2011.
[13] L. Liu, V. Saripalli, V. Narayanan, and S. Datta, "Device circuit co-design using classical and nonclassical III-V multigate quantum-well FETs (MuQFETs)," in *Proc. IEEE IEDM*, Dec. 2011, pp. 4.5.1–4.5.4.
[14] K. Matsumoto, M. Ishii, K. Segawa, Y. Oka, B. J. Vartanian, and J. S. Harris, "Room temperature operation of a single-electron transistor made by the scanning tunneling microscope nanooxidation process for the TiOx/Ti System," *Appl. Phys. Lett.*, vol. 68, no. 1, pp. 34–36, 1996.
[15] H. W. Ch. Postma, T. Teepen, Z. Yao, M. Grifoni, and C. Dekker, "Carbon nanotube single-electron transistors at room temperature," *Science*, vol. 293, pp. 76–79, Jul. 2001.
[16] P. Rashinkar, P. Paterson, and L. Singh, *System-on-a-Chip Verification: Methodology and Techniques*, Norwell, MA, USA: Kluwer Academic Publishers, 2000.
[17] V. Saripalli, L. Liu, S. Datta, and V. Narayanan, "Energy-delay performance of nanoscale transistors exhibiting single electron behavior and associated logic circuits," *J. Low Power Electron.*, vol. 6, pp. 415–428, Oct. 2010.
[18] S. J. Shin, C. S. Jung, B. J. Park, T. K. Yoon, J. J. Lee, S. J. Kim, J. B. Choi, Y. Takahashi, and D. G. Hasko, "Si-based ultrasmall multiswitching single-electron transistor operating at room-temperature," *Appl. Phys. Lett.*, vol. 97, no. 10, pp. 103101-1–103101-3, 2010.
[19] G. S. Tseitin, "On the complexity of derivation in propositional calculus," *Stud. Constr. Math. and Math. Logic*, 1968, pp. 115–125.
[20] Y. T. Tan, T. Kamiya, Z. A. K. Durrani, and H. Ahmed, "Room temperature nanocrystalline silicon single-electron transistors," *J. Appl. Phys.*, vol. 94, pp. 633–637, Jul. 2003.
[21] K. Uchida, J. Koga, R. Ohba, and A. Toriumi, "Programmable single-electron transistor logic for future low-power intelligent LSI: Proposal and room-temperature operation," *IEEE Trans. Electron. Devices*, vol. 50, no. 7, pp. 1623–1630, Jul. 2003.
[22] S. Yang, "Logic synthesis and optimization benchmarks, version 3.0," Tech. Rep., Triangle Park, NC, USA: Microelectron. Center North Carolina, Jan. 1991.
[23] L. Zhuang, L. Guo, and S. Y. Chou, "Silicon single-electron quantum-dot transistor switch operating at room temperature," *Appl. Phys. Lett.*, vol. 72, no. 10, pp. 1205–1207, Mar. 1998.
[24] [Online]. Available: http://minisat.se/
[25] [Online]. Available: http://www.princeton.edu/∼chaff/zchaff.html

Yung-Chih Chen 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 was a Visiting Student with the Department of Computer Science and Engineering, Pennsylvania State University, University Park, PA, USA, from June 2010 to March 2011. Then, he joined the Department of Electronic Engineering, Chung Yuan Christian University, Taoyuan, Taiwan, where he was an Assistant Professor from 2011 to 2012. He is currently an Assistant Professor with the Department of Computer Science and Engineering, Yuan Ze University, Taoyuan. His current research interests include logic synthesis, design verification, and design automation for emerging technologies.

Chun-Yao Wang (S'00–M'03) 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 an Associate Professor. In 2010, he was a Visiting Professor with the Department of Computer Science and Engineering, Pennsylvania State University, University Park, PA, USA. He holds six patents. His current research interests include logic synthesis, optimization, and verification for very large-scale integrated/system-on-chip designs and emerging technologies. He has published over 50 technical papers in these areas.

Dr. Wang received the Distinguished Teaching Award from the College of Electrical Engineering and Computer Science, National Tsing Hua University, in 2007 and 2008. In 2009, he was the recipient of the Excellent Young Electrical Engineer Medal from the Chinese Institute of Electrical Engineering, Taipei. In 2011, he was granted the Excellent Young Scholar Research Project from National Science Council, Taiwan. Two of his 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, respectively. He has served as Technical Program Committee Member and Organizing Committee Member of several conferences, including Design, Automation and Test in Europe, Asia and South Pacific Design Automation Conference, Great Lake Symposium on Very Large Scale Integration, and Embedded Systems Week.

Ching-Yi Huang received the B.S. degree from the Department of Computer Science, National Tsing Hua University, Hsinchu, Taiwan, in 2009, where he is currently pursuing the Ph.D. degree in the Department of Computer Science.

His current research interests include logic synthesis and design verification.