Dynamic Diagnosis for Defective Reconfigurable Single-Electron Transistor Arrays

Yun-Jui Li, Ching-Yi Huang, Chia-Cheng Wu, Yung-Chih Chen, Chun-Yao Wang, *Member, IEEE*, Suman Datta, *Fellow, IEEE*, and Vijaykrishnan Narayanan, *Fellow, IEEE*

***Abstract*— Single-electron transistor (SET) at room temperature has been demonstrated as a promising device for extending Moore's law due to its ultralow-power consumption. Previous works proposed mapping approaches to implement Boolean functions on SET arrays. However, these approaches were based on an ideal assumption that the SET arrays are defect-free. Recently, a diagnosis method was proposed targeting at defective SET arrays. However, the approach was static, such that the performance is inefficient. As a result, in this paper, we propose a dynamic diagnosis approach that can efficiently identify the locations and the types of the defects in the SET arrays. The experimental results show that the proposed dynamic diagnosis approach can achieve the same results as the previous work with much less CPU time on a set of benchmarks. Furthermore, the proposed method spent a few seconds while the previous work exceeded the CPU time limit of 3600 s on some benchmarks.**

***Index Terms*— Diagnosis, dynamic, optimization, single-electron transistor (SET) array.**

I. Introduction

THE increasing power consumption is one of the primary bottlenecks to extend Moore's law. Many low-power devices have been proposed to overcome this issue. Among these devices, some demonstrations of single-electron transistors (SETs) operating at room temperature have shown SET to be a promising candidate to extend Moore's law [18]–[20], [23].

Since only a few electrons are involved in the switching operation for SETs, the low transconductance is a major issue, such that CMOS-based logic architecture is not suitable for SETs. Thus, a binary decision diagram (BDD)-based architecture was proposed in [1] to implement logic functions on SETs. Furthermore, the BDD of a Boolean function can be used to map onto an SET array [8], [13], [22].

However, if there is any defect occurring on the SET devices or the nanowire segments in the SET array, the SET array fails to achieve its function. This causes a low yield of SET arrays due to the high defect rate of nanodevices and nanowires. To improve the reliability of SET arrays, reconfigurable SET architecture was proposed in [7]. This success promotes the development of automation tools for the synthesis and verification of SET arrays. Chen *et al.* [2] proposed the first automatic synthesis approach in the literature. After that, much research [3]–[6], [16], [17], [22] focused on minimizing the area of the mapped SET arrays.

Although the above-mentioned mapping methods for area minimization were effective, they did not consider the defects, which could influence the correctness of the implemented function, within the SET arrays. Thus, a defect-aware mapping algorithm was proposed in [11], which relied on the defect information to detour or reuse the defects successfully while mapping. That work assumed that designers have known the locations and the types of all defects in the SET arrays before mapping. Unfortunately, it is not the case for defective SET arrays in practice. Thus, it is important to have a diagnosis method to identify the defects within SET arrays for succeeding mapping algorithm.

As a result, the previous work [12] proposed the first diagnosis algorithm to identify the defects in a reconfigurable SET array statically. The diagnosis algorithm exploits a *diagnosis sequence* consisting of input patterns and configurations to achieve this goal. However, the proposed algorithm is static, since: 1) the diagnosis sequence is fixed regardless the locations and the types of defects and 2) the diagnosis process will not be terminated until the whole SET array is traversed completely. Although this static diagnosis approach is effective and easy-to-understand, it is inefficient. This is because it spent a large amount of redundant efforts on the edges that have been diagnosed. Thus, in this paper, we propose a dynamic approach that exploits the responses obtained in the diagnosis process to adjust the succeeding diagnosis sequences.

The main contributions of this paper are twofold.

1) This is the first work using the dynamic approach to diagnose defective SET arrays.
2) The efficiency of the proposed approach is significantly elevated as compared with the state of the art.

The rest of this paper is organized as follows. Section II describes the background of SET arrays. Section III presents the proposed diagnosis approach for defective SET arrays.

Manuscript received May 17, 2016; revised August 22, 2016 and October 24, 2016; accepted November 27, 2016. Date of publication January 9, 2017; date of current version March 20, 2017. This work was supported by the Ministry of Science and Technology of Taiwan under Grant MOST 103-2221-E-007-125-MY3, Grant MOST 103-2221-E-155-069, Grant MOST 104-2220-E-155-001, Grant NSC 100-2628-E-007-031-MY3, Grant NSC 101-2221-E-155-077, Grant NSC 101-2628-E-007-005, Grant NSC 102-2221-E-007-140-MY3, and Grant NSC 102-2221-E-155-087.

Y.-J. Li, C.-Y. Huang, C.-C. Wu, and C.-Y. Wang are with the Department of Computer Science, National Tsing Hua University, Hsinchu 30013, Taiwan (e-mail: abc61219@yahoo.com.tw; s986516@m98.nthu.edu.tw; orange172839@yahoo.com.tw; wcyao@cs.nthu.edu.tw).

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

S. Datta is with the College of Engineering, University of Notre Dame, Notre Dame, IN 46556 USA (e-mail: sdatta@nd.edu).

V. Narayanan is with the Department of Computer Science and Engineering, Pennsylvania State University, University Park, PA 16802 USA (e-mail: vijay@cse.psu.edu).

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/TVLSI.2016.2639533

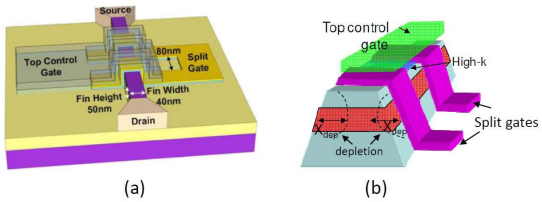


Fig. 1. (a) Structure of a reconfigurable SET device [12]. (b) Formulations of wrap-around Schottky split gates and the top control gate [7].

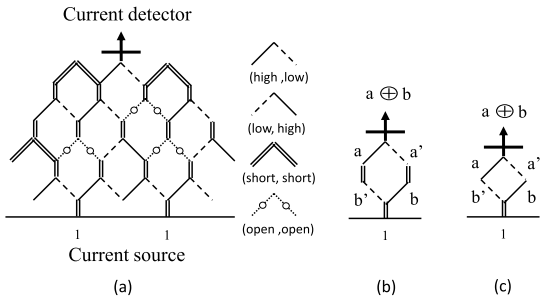


Fig. 2. (a) SET array. (b) Example of $a \oplus b$. (c) Simplified diamond-shaped network of $a \oplus b$ [2].

Section IV shows the experimental results. Section V concludes this paper.

II. Preliminaries

A. Reconfigurable SET Array

The structure of a reconfigurable SET device is shown in Fig. 1, where a pair of Schottky gates, called split gates, are wrapped around the fin that connects the source and the drain, and the top control gate is built upon the splits gates. By providing the split gates a voltage bias, the SET can be set in three modes of operations: 1) active; 2) open; and 3) short modes. In the active mode, the split gate bias is adjusted to make the tunneling resistance of the source and drain junctions to exceed the resistance quantum, but is still low enough to permit efficient tunneling. Then, the voltage bias applied from the top control gate (input signal) controls the dot potential to block or permit electrons tunneling. In the open mode, the split gate bias is set to a sufficiently negative value to let the depletion regions from both sides to encroach and pinch off the nanodot island completely. Finally, in the short mode, a large enough positive split gate bias is applied, so that the tunnel junctions become almost transparent and the tunneling resistance is significantly reduced. In other words, the device behaves like a near ohmic conductor.

A reconfigurable SET array can be represented as a hexagonal network, as shown in Fig. 2(a). There are current sources and a current detector at the bottom and the top of the reconfigurable SET array, respectively. The current detector is considered as the output of the SET array. When the current comes from a current source and reaches the detector, the output value is 1; otherwise, the output is 0. Each sloping edge in the SET array represents an SET device, which can be configured as *active high, active low, short*, or *open*. The current passes through an *active high* (or simply *high*) edge when the input to the edge is logic 1; the current is blocked when the input to the edge is logic 0, and vice versa for *active low* (*low*). A *short* (*open*) edge represents electrical short (open). A node device is composed of a pair of sloping edges, and the node devices on the same row share the same input. There are connections between the current sources and the SET array, which can be configured as *short* or *open* as well. The inputs among the rows constitute an input pattern. When applying an input pattern to an SET array, if there exists a path that can transport the electrons from the current source to the detector, the path is a conducting path under this input pattern.

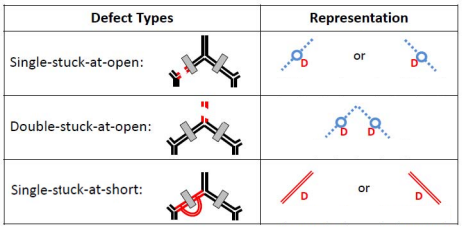


Fig. 3. Fabric representations of the three types of failures [11].

For example, Fig. 2(b) shows an implementation of $a \oplus b$ on an SET array. The output value will be 1 when the input pattern is either ($ab = 01$) (via the right path) or ($ab = 10$) (via the left path). The other input patterns ($ab = 00$ or 11) will lead the output value to be 0. Fig. 2(c) is a simplified version of Fig. 2(b) by removing the vertical edges of the hexagons, since they are electrically *short*. In the rest of this paper, only the sloping edges will be shown in the SET arrays.

B. Symmetric Fabric Constraint

To reduce the number of input wires that are used to configure the node devices in the SET array, the *symmetric fabric constraint* [7] is imposed in all the related works [2]–[6], [11], [16], [17], [22]. The *symmetric fabric constraint* limits the configuration of a node device can only be one of *(high, low)*, *(low, high)*, *(short, short)*, or *(open, open)*, as shown in Fig. 2(a). Furthermore, the configuration of *(high, low)* and *(low, high)* cannot appear in the same row of a reconfigurable SET array simultaneously.

To simplify the description about our diagnosis method, without loss of generality, we only use three types of configurations in this paper, which are *(high, low)*, *(short, short)*, and *(open, open)*. Therefore, when we set a node to the active mode, it means that the node is configured as *(high, low)*.

C. Defect Model

A defect model for defective SET arrays was proposed in the previous work [11], which considers three types of defects, i.e., *single-stuck-at-open*, *double-stuck-at-open*, and *single-stuck-at-short*, as shown in Fig. 3. In this paper, we adopt the same defect model [11] in our diagnosis approach as the state of the art [12] did. Furthermore, the *single-stuck-at-open* defect and the *double-stuck-at-open* defect are categorized as open defects while the *single-stuck-at-short* defect is categorized as a short defect. When an edge of a node is defective, its behavior does not change with a configuration. It means that an open-defect

edge is always *open* while a short-defect edge is always *short*. Finally, the connections between the SET array and the current source could also be defective.

III. Diagnosis Approach

A. Defect Distribution

The defect rate of the SET devices could be as higher as $2\% = 20\,000$ ppm since it is more vulnerable than MOS [12]. This defect rate means that there are two defects among 100 nodes on average, which is quite sparse. Thus, we also adopt the assumption of defect distribution proposed in [12] in this paper. First, open defects and short defects do not occur in a node device simultaneously. Second, any two defective nodes, which contain at least one defective edge, are not adjacent to each other. Therefore, if a node device is identified as having a defective edge, its six adjacent nodes are assumed to be defect-free.

B. Overview

The diagnosis for reconfigurable SET arrays is different from that for traditional Boolean circuits [10], [14], [15], [21] due to functional reconfigurability of SET arrays. The SET arrays exploit both SET node configurations and input patterns to represent a circuit's functionality. Thus, the diagnosis process can utilize this information to identify the defects. The node configuration is a setting of node-under-diagnosis, which is one of *(high, low), (short, short)*, or *(open, open)*. Since the total amount of node configurations and input patterns, which is named *diagnosis cost*, is strongly correlated with the time spent during diagnosis, our diagnosis approach will also minimize this cost by determining a good diagnosis sequence. The problem formulation of this paper is as follows.

Problem formulation: Given a defective reconfigurable SET array, we would like to identify the locations and the types of all the defects by determining the node configurations and the input patterns with the minimized diagnosis cost.

We utilize two ideas to identify the defects.

1) If a configured path is conducting under an input pattern, the edges in the path do not suffer from any open defects. Hence, each edge of this path will be marked as having a nonopen defect.
2) After having a conducting path without open defects, we can diagnose whether a short defect occurs on an edge by changing the configuration of the corresponding node to a new configuration (open, open). If the path is still conducting after this new configuration, the corresponding edge will be marked as a short defect. Otherwise, it is defect-free due to the absence of open defects and short defects.

The proposed diagnosis approach contains two stages:

1) diagnosis with conducting paths;
2) diagnosis for remaining short defects.

The main concept of our method is to exploit the edges whose statuses have been identified, i.e., short, open, or defect-free, to diagnose other adjacent edges dynamically. First, we try to find a conducting path. If we can find one, we replace some edges in the path with other adjacent edges-under-diagnosis. Therefore, when observing a defect effect at the current detector, we can realize that the defect must occur among the replaced edges-under-diagnosis. However, if we cannot find a conducting path, our approach will fail. We will see the success rate of this operation in the experimental results. Second, we diagnose short defects and open defects on the edges of this conducting path and its neighboring edges. This is because the both ideas mentioned earlier are based on conducting paths. Finally, we diagnose the remaining edges, which are left from the previous diagnosis processes, by creating conducting paths using the identified defective or defect-free edges.

Before introducing the proposed approach, we use an example to illustrate the difference between the static approach in [12] and the dynamic one in this paper. In Fig. 4, assume that we only diagnose open defects for the edges in the paths rooted from the node (0, 0) in a 4×8 SET array for brevity. The height of SET array is 4; hence, the total number of paths rooted from (0, 0) is $2^4 = 16$. Originally, the diagnosis process for open defects in [12] will traverse all these paths. However, only the input patterns from 0001 to 1110 (in binary) are applied, since the patterns 0000 and 1111 are corresponding to the undiagnosable edges, which will be explained later. Fig. 4(a)–(d) shows the first to fourth path and the corresponding input patterns in the SET array. Next, we calculate the number of configurations from one path to another path. For example, the first path needs five configurations, including the connection to the current source, as shown in Fig. 4(a). It needs two additional configurations to change paths from Fig. 4(a) and (b), where the nodes at (1, 3) and (3, 3) are reconfigured. Similarly, it needs two additional configurations to change from Fig. 4(b) and (c) by reconfiguring the connections to the current source. In summary, 39 configurations and 14 input patterns are required in the static diagnosis method.

On the contrary, the diagnosis processes for the open defects in this paper are shown in Fig. 4(a) and (e)–(l). The numbers of configurations from one figure to the next figure are 2, 3, 3, 2, 2, 3, 2, 2. Therefore, the numbers of required configurations and input patterns are $24(5+19)$ and 9,[1] respectively. Furthermore, the number of configurations and input patterns required in the static approach grows exponentially with respect to the

[1]The first path needs five configurations, including the nodes at (0, 0), (1, 1), (2, 2), and (3, 3), and the connection to the current source, as shown in Fig. 4(a). It also needs two additional configurations to change paths from Fig. 4(a)–(e), where the nodes at (1, 1) and $(-1, 1)$ are reconfigured. It also needs three additional configurations to change paths from Fig. 4(e) and (f), where the nodes at (1, 1), (2, 2), and (0, 2) are reconfigured. Then, it needs another three additional configurations to change paths from Fig. 4(f) and (g), where the nodes at (2, 2), (3, 3), and (1, 3) are reconfigured. Next, it needs two additional configurations to change paths from Fig. 4(g) and (h), where the nodes at (0, 2) and $(-2, 2)$ are reconfigured. After that, it needs two additional configurations to change paths from Fig. 4(h) and (i), where two connections to the current source are reconfigured. Similarly, it needs three additional configurations to change paths from Fig. 4(i) and (j), where the nodes at (0, 2), (1, 3), and $(-1, 3)$ are reconfigured. Finally, it needs two additional configurations to change paths from Fig. 4(j) and (k), and two additional configurations to change paths from Fig. 4(k) and (l). Thus, the total number of required configurations is $5+2+3+3+2+2+3+2+2 = 24$. Furthermore, each path needs a corresponding input pattern for simulation. Therefore, the number of required input patterns is 9.

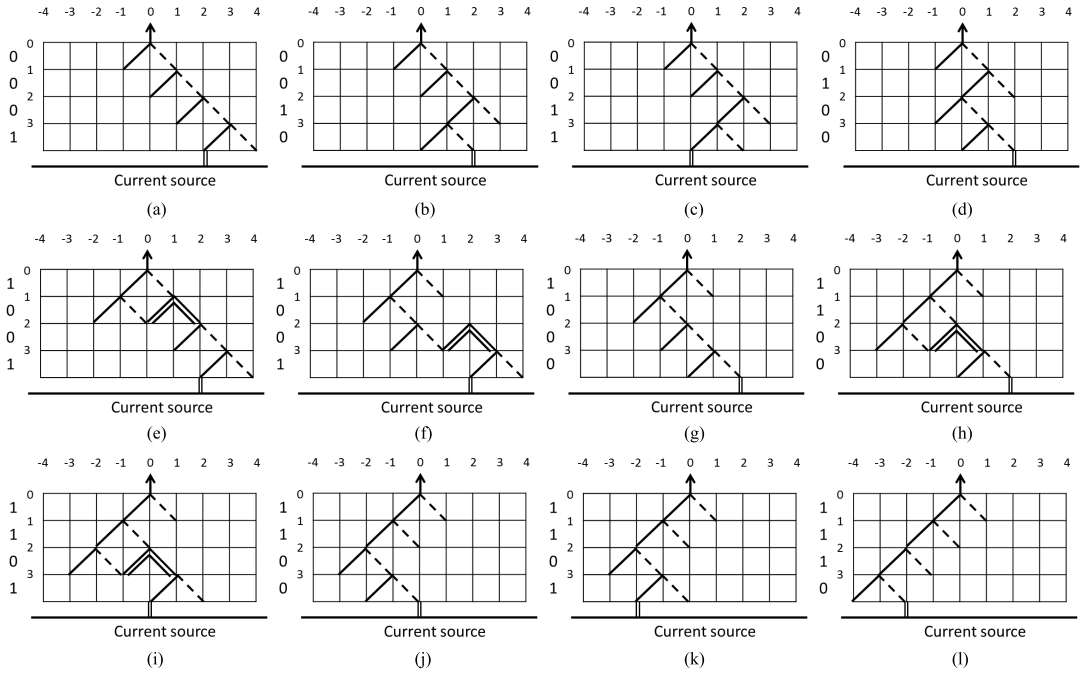


Fig. 4. Example of the open-defects diagnosis sequences by the static approach [12] and the proposed dynamic approach. (a)–(d) The first to fourth path and the corresponding input patterns of static defect diagnosis approach in the SET array. (e)–(l) The paths and the corresponding input patterns of dynamic defect diagnosis approach in the SET array.

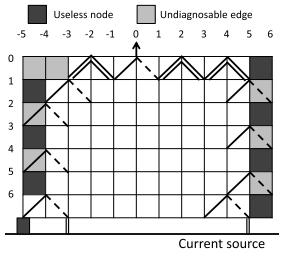


Fig. 5. Example of undiagnosable edges and useless nodes in an SET array.

height of SET array while the dynamic approach does not. We will elaborate this in the succeeding paragraphs.

Since an SET node is composed of a pair of edges, the boundary node of an SET array that is with only one edge is considered as a *useless node*, and will be discarded. Furthermore, there are some edges that cannot be involved in a conducting path due to no nodes below them for building a complete path. These edges are named *undiagnosable edges*. For example, Fig. 5 shows the undiagnosable edges, which are represented in gray, in an SET array. They are unable to build a conducting path, since they are connected to useless nodes, which are represented in black. Since the undiagnosable edges cannot be further utilized to build paths in the mapping approaches either, we ignore them and the useless node edges in the calculation of the diagnosis coverage. Furthermore, these undiagnosable edges and useless nodes are marked as open defects after identifying all the edges on the SET array.

The details of the above-mentioned stages will be discussed in Sections III-C and III-D.

C. Diagnosis With Conducting Paths

Diagnosis with conducting paths is the first stage of the proposed approach. It contains three steps and each step will be explained in Sections III-C1–III-C3. The second and third steps will be executed iteratively until this stage terminates.

1) Finding a Conducting Path: In the beginning, we configure all the nodes in the SET array as *(open, open)* for reset and mark all the edges as nonidentified. Then, we use a trial-and-error method to find a conducting path. Since the defects in an SET array are sparse under the assumptions of defect rate and defect distribution, finding a conducting path is quite possible after a few trials. We will see this result in the experiments. Thus, we randomly configure a path from the current detector to the current source and apply the corresponding input pattern to see whether the output is 1. If the output is 1, the path is conducting and is selected as the *baseline path*; otherwise, we configure other paths. This process is repeated until a conducting path is found. For example, Fig. 6(a) is a defect map of an SET array, and the locations and the types of defects are unknown originally. Fig. 6(b) shows a trial nonconducting path due to a *double-stuck-at-open* defect at $(-1, 1)$ with the corresponding input pattern 1111011. Fig. 6(c) shows a found conducting path, which serves as the baseline path. In the rest of this paper, the nodes that do not show their configurations represent the *(open, open)* configurations for brevity. After finding the baseline path, we mark all the edges on this path as *nonopen defects*. This baseline path will be referred further in the next step.

2) Applying Patterns for Short Defects: We apply additional input patterns to the baseline path again for detecting short defects on the nonopen edges under the same configuration. An additional input pattern can be obtained by flipping

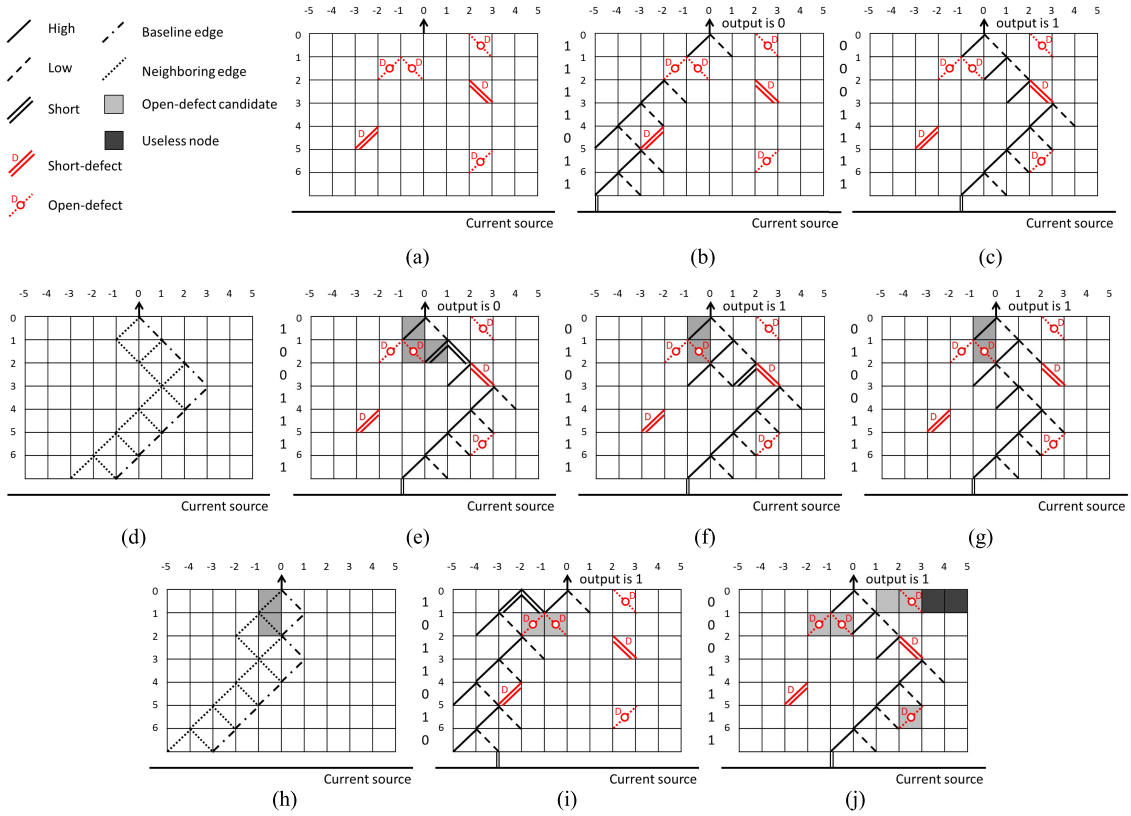


Fig. 6. Example of diagnosis with conducting paths. (a) Defect map of the reconfigurable SET array. (b) Nonconducting path. (c) Conducting path as the first baseline path. (d) First baseline path and its neighboring edges. (e)–(g) Diagnosing the neighboring edges of the first baseline path. (h) New neighboring edges and the open-defect candidates. (i) Baseline path involving the expansion node at $(-2, 0)$. (j) Diagnosing the right neighboring edges of the first baseline path.

the corresponding input bits of the edges-under-diagnosis. If the output is 0 after applying the additional pattern, the edges-under-diagnosis are not short. Hence, we can conclude that these nonopen edges are not short either, i.e., they are defect-free. On the other hand, if the output is 1 and only one edge-under-diagnosis is involved, the edge is a short-defect edge. However, if more than one edge-under-diagnosis are involved, we cannot determine which edge has a short defect. Hence, the statuses of these edges will be determined in the second stage of diagnosis for remaining short defects. Furthermore, in the whole diagnosis process, if one edge has been diagnosed already, it will be skipped from the succeeding process in this dynamic approach.

For example, the original input pattern for the first baseline path in Fig. 6(c) is 0001111. Since the edges on the baseline path are nonopen edges, we further identify the short defects on the path. We flip the first bit of the input pattern, i.e., from 0001111 to 1001111, and the output becomes 0. Thus, we can realize that the right edge of the node at (0, 0) is not a short edge. Since this edge has been identified as a nonopen edge earlier,[2] it is a defect-free edge. On the contrary, when we apply the pattern 0011111 to the baseline path in Fig. 6(c), the output is still 1, which means that a short defect occurs. However, we cannot tell whether the right edge of the node at (2, 2) or the right edge of the node at (1, 3) is a short-defect edge. We just leave this determination to the stage of diagnosis for remaining short defects. Note that we also exploit the assumption of the defect distribution in this paper to identify defects, i.e., when we know an edge is a defective edge, all its six adjacent nodes are defect-free nodes.

3) Diagnosis for the Neighboring Edges: Without loss of generality, the step of diagnosis for the neighboring edges starts from the left neighboring edges of the baseline path. For example, Fig. 6(d) shows the left neighboring edges of the baseline path, which are represented in dotted lines. In our approach, the diagnosis order for the neighboring edges starts from the top level to the bottom one.

To diagnose the neighboring edges, we take the baseline path as a main trunk and create branches to cover the neighboring edges. Therefore, we modify the configurations to replace some edges in the baseline path with the neighboring edges-under-diagnosis temporarily, and change the input pattern with respect to the new path. If the output of the new path is 1, the neighboring edges-under-diagnosis are marked as nonopen. Conversely, if the output is 0, the newly added neighboring edges are considered as open-defect candidates. The reason that they are just the candidates is because the number of newly added neighboring edges is greater than one.

Note that if the output is 0 after diagnosing a set of neighboring edges, we have to recover the baseline path. However, if the output is 1, we set up a new baseline path by considering the previous neighboring edges as a part of the new baseline path. Therefore, the baseline path is not fixed during the diagnosis process for reducing the diagnosis

[2]The edge is on the baseline path.

cost. Also, if an open-defect candidate is not adjacent to other candidates, the open-defect candidate is updated as an open-defect edge due to uniqueness; otherwise, we determine its status later.

We take Fig. 6(e)–(g) as examples to show the diagnosis process for a set of neighboring edges. First, in Fig. 6(e), we reconfigure the nodes at $(-1, 1)$ and $(1, 1)$ as *(high, low)* and *(short, short)*, respectively, to diagnose the left edge of the node at (0, 0), the right edge of the node at $(-1, 1)$, and the left edge of the node at (1, 1). We also change the input pattern from 0001111 to 1001111, as shown in Fig. 6(e). After applying this input pattern and having the output value of 0, we mark these three edges-under-diagnosis as the open-defect candidates. Second, we recover the baseline path by configuring *(open, open)* and *(high, low)* at the nodes $(-1, 1)$ and $(1, 1)$. Then, in Fig. 6(f), we configure the nodes at (0, 2) and (2, 2) as *(high, low)* and *(short, short)*, respectively, to diagnose the left edge of the node at (1, 1), the right edge of the node at (0, 2), and the left edge of the node at (2, 2). The input pattern is modified from 0001111[3] to 0101111 accordingly, as shown in Fig. 6(f). After applying this input pattern and having the output value of 1, we mark these three edges-under-diagnosis as nonopen edges. Note that the left edge of node at (1, 1) is removed from the open-defect candidates, since it is identified as a nonopen edge currently. Next, since the output value in Fig. 6(f) is 1, we reuse the configurations in Fig. 6(f) when we further diagnose the right edge of the node at (1, 3), which is the neighboring edge of the first baseline path. Therefore, we reconfigure both the nodes at (2, 2) and (3, 3) as *(open, open)* to change the baseline path. Then, we reconfigure the node at (1, 3) as *(high, low)* and apply the input pattern 0100111. The output value is 1, as shown in Fig. 6(g), which indicates that the right edge of node at (1, 3) is a nonopen edge.

To diagnose the defects dynamically for efficiency elevation, the baseline paths are changed during diagnosis if applicable. A path without open-defects and open-defect candidates may be a baseline path. For example, Fig. 6(h) shows a new baseline path, which detours the open-defect candidates (in gray). The new left neighboring edges are shown as well in Fig. 6(h). Then, we perform the same process to diagnose these left neighboring edges until reaching the undiagnosable edges or the useless nodes.

Note that when creating a new baseline path, we may configure some nodes in the first row as *(short, short)* to be the expansion nodes, as shown in Fig. 6(i). The creation of this new baseline path in Fig. 6(i) also confirms that the left edge of the node at (0, 0) is not an open defect. As a result, the left edge of the node $(-1, 1)$ will be recognized as an open-defect edge, since it has been diagnosed as an open-defect candidate in the previous diagnosis process. Thus, we update the status of this edge by removing it from the open-defect candidates.

When all the neighboring edges of the original baseline path are diagnosed, the updated baseline path may be different from the original one. We can consider this updated baseline path as a new baseline path coming from the step of *finding a conducting path*. Thus, we apply the second and third steps to it as well. The diagnosis process continues until reaching the undiagnosable edges or the useless nodes.

Next, we diagnose the right-hand side edges of the original baseline path using the same steps as earlier, and the result is shown in Fig. 6(j). For the nodes in the first row excluding the one at (0, 0), having a *single-stuck-at-open* defect is equivalent to having a *double-stuck-at-open* defect due to the same defect effect. The situation is shown in Fig. 6(j). That is, the left edge of the node at (2, 0) is actually defect-free, but we mark it as an open-defect candidate, since its defect effect is the same. Furthermore, if a node in the first row is blocked by an open defect, it becomes useless, since it cannot be used as an expansion node anymore. Therefore, we will mark it as a useless node without diagnosing it, as Fig. 6(j) shows.

Finally, the remaining open-defect candidates are updated as open defects before closing this stage of *diagnosis with conducting paths*.

D. Diagnosis for Remaining Short Defects

As mentioned in Section III-B, short defects are identified by reconfiguring a node as *(open, open)* in a conducting path. If the path is still conducting after the reconfiguration, the node has a short defect; otherwise, it has no short defect. Since the locations of open defects have been identified in the last stage, it is much easier to build a conducting path in this stage. To diagnose the remaining short defects systematically in this stage, the conducting baseline paths are built columnwise from the middle to the left, and then to the right of an SET array. Of course, the open-defect edges are still detoured while building the baseline paths.

For example in Fig. 7(a), in the construction of the first baseline path in $C1$, the nodes whose x-coordinates are 0 or -1 are involved. However, there is an open-defect edge in $C1$; hence, we detour it, such that the first baseline path is as shown in Fig. 7(a). After configuring the first baseline path, the diagnosis sequence starts from the top edges to the bottom ones within the baseline path. Furthermore, if an edge in the baseline path has been diagnosed as a short defect or defect-free, we skip it in the diagnosis process.

For diagnosing the remaining short defects, we change the configurations of the edges-under-diagnosis from the *(high, low)* to *(open, open)*, and apply the corresponding input pattern. If the output is 0, the edge-under-diagnosis is a defect-free edge due to the absence of open defects and short defects. On the contrary, if the output is 1, the edge-under-diagnosis is a short-defect edge.

Before diagnosing the next edge, we have to recover the baseline path by reconfiguring the node to *(high, low)*. Note that for the nodes in the first row, only the node at (0, 0) will be diagnosed for short defects. This is because the other nodes in the first row are only used for expansions, which are exactly configured as *(short, short)*. In other words, if short defects occur in the first row, but are not at the node of (0, 0), they are harmless. These short defects are considered as *don't-care defects*, and they will be ignored in the calculation of the *coverage*.

[3]This is the input pattern for the baseline path.

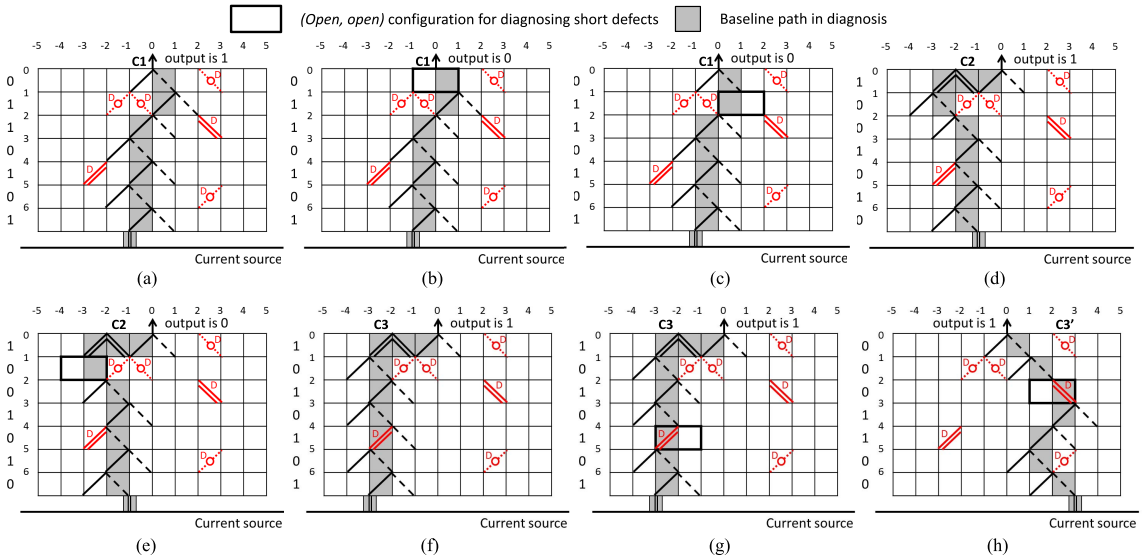


Fig. 7. Example of diagnosis for remaining short defects. (a) First baseline path, which detours a *double-stuck-at-open* defect. (b) and (c) Diagnosing short defects of the edges in the first baseline path. (d) Second baseline path, which detours a *double-stuck-at-open* defect. (e) Diagnosing short defects of the right edge of the node at $(-3, 1)$, which is in the second baseline path. (f) Third baseline path. (g) Identifying the short-defect edge at the left edge of the node at $(-2, 4)$. (h) Identifying the short-defect edge at the right edge of the node at $(2, 2)$.

For example, Fig. 7(b) and (c) shows the process of diagnosing the first two edges in the first baseline path, respectively. As shown in Fig. 7(b), we reconfigure the node at (0, 0) as *(open, open)* and observe the output value of 0 with the corresponding input pattern. Thus, the right edge of the node at (0, 0) is defect-free. Then, we recover the first baseline path by reconfiguring the node at (0, 0) to *(high, low)*. Next, we reconfigure the node at (1, 1) to *(open, open)* and observe the output value of 0, as shown in Fig. 7(c). Therefore, the left edge of the node at (1, 1) is defect-free as well.

After dealing with the edges in the first baseline path, we build the second baseline path in $C2$. The open defects still have to be detoured if and only if they block the baseline path. For example, Fig. 7(d) shows the second baseline path. Note that the different edges of one node could be used to build different baseline paths. Hence, such a node does not need to be reconfigured, whereas the corresponding input pattern has to be changed. For example, two different baseline paths, as shown in Fig. 7(d) and (f), reuse the configurations of the nodes at $(-2, 2)$, $(-2, 4)$, and $(-2, 6)$ and the expansion node at $(-2, 0)$. Fig. 7(g) shows that the left edge of the node at $(-2, 4)$ is diagnosed as a short-defect edge, since the output is 1 under the *(open, open)* configuration at $(-2, 4)$. Similarly, the right edge of the node at (2, 2) is diagnosed as a short-defect edge in $C3'$, as shown in Fig. 7(h).

When all the columns have been diagnosed, all the short defects are identified. Finally, we mark the undiagnosable edges and useless nodes as open defects. Combining with the identified defects in the first and the second stage, the diagnosis of the SET array is finished.

E. Overall Flow

Fig. 8 shows the flowchart of the proposed dynamic diagnosis approach. Given an SET array, we first reset the configurations of all the nodes as *(open, open)*. Second, we try to find a conducting path as the baseline path. If the baseline path is found, we apply the input patterns to diagnose short-defect and open-defect edges in the baseline path as well as its left neighboring edges; otherwise, we discard the SET array. After reaching the undiagnosable edges or the useless nodes, we apply the same process to the right neighboring edges of the original baseline path. Then, we diagnose the remaining short defects. Finally, we mark the undiagnosable edges and useless nodes as open defects, and report the locations and the types of the identified defects as a defect map.

F. Time Complexity Analysis

To derive the time complexity of the proposed approach, we calculate the time complexity of each step, and then sum them up to obtain the overall result.

First, the time complexity of the first step in the first stage, which is finding a conducting path, is O(1), since we set the trial limit as a constant in the algorithm.

Second, we calculate the time complexity of the second stepapplying patterns for short defects and the third step-diagnosis for the neighboring edges simultaneously. We put these two steps together, because they are executed iteratively until the termination of this stage. Since these two steps will diagnose several baseline paths individually, we split the calculation into single baseline path and multiply the result of a single baseline path by the iteration times. In the second step, for each baseline path, we need to apply an input pattern for diagnosing every edge of the baseline path. Thus, the number of the required input patterns is equal to the height of the SET array. In the third step, for each edge of each baseline path, we diagnose the neighboring edges on the same row. Hence, the number of the required input patterns is equal to the height of the SET array as well. Then, these results need to be multiplied by the number of baseline paths in the SET array. Also, the number of baseline paths is proportional to the width of the SET array. As the result, the time complexity of the second and the third steps is O(height $\times$ width), where

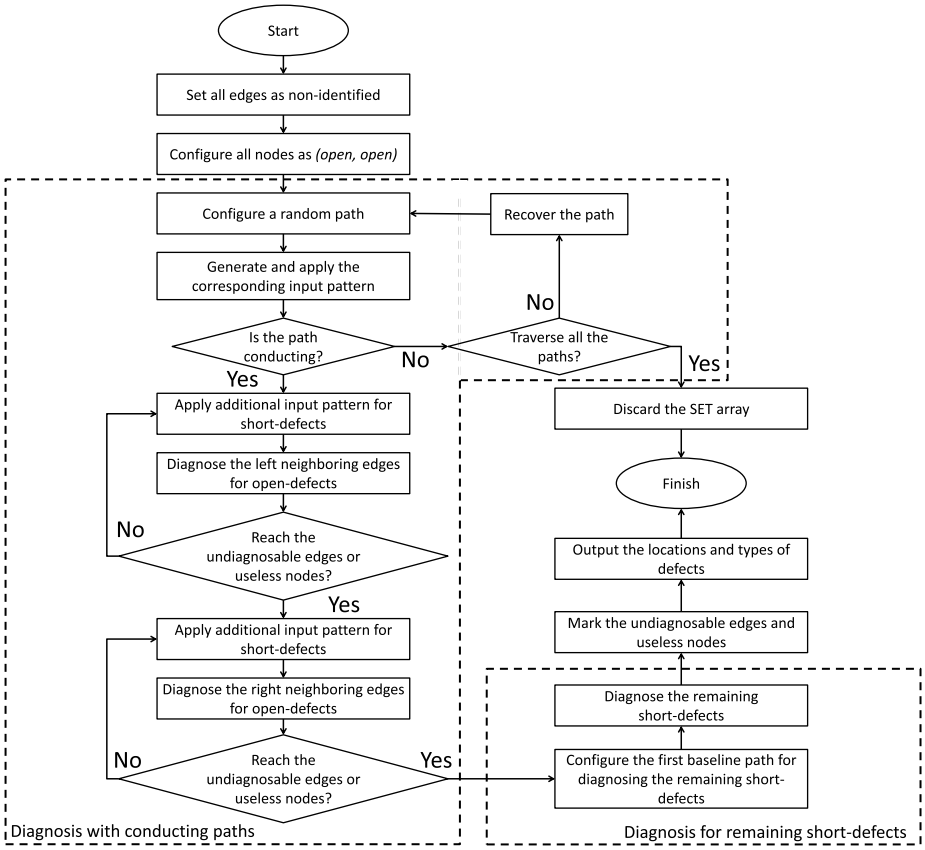


Fig. 8. Flowchart of the proposed dynamic diagnosis approach.

height and width represent the height and the width of an SET array.

Finally, the time complexity of the second stage, which is diagnosis for remaining short defects, is O(height $\times$ width) as well. The analysis of this stage is similar to the second and the third steps in the first stage.

We sum up these three results mentioned earlier to derive the overall time complexity of the proposed approach

$$\begin{aligned} &\mathrm{O}(1) + \mathrm{O}(\text{height} \times \text{width}) + \mathrm{O}(\text{height} \times \text{width}) \\ &\quad = \mathrm{O}(\text{height} \times \text{width}). \end{aligned}$$

Thus, the time complexity of the proposed diagnosis approach is O(height $\times$ width), which can also be represented as O(|SET node|).

IV. Experimental Results

We implemented the proposed algorithm in C++ language and conducted the experiments on an Intel Xeon X5570 2.93-GHz CentOS 5.1 platform with 48-GB memory. We conducted three experiments in this paper with different sizes of defective SET arrays. The first experiment is to demonstrate the success rate of finding a conducting path in a defective SET array. The second experiment is to show the comparison between the state of the art and our approach on the efficiency of diagnosis process. The third experiment is to show the number of undiagnosable edges and useless nodes in the SET array and the number of defects in them. In the experiments, the defect rates of the three defect models, *single-stuck-at-open, single-stuck-at-short*, and *double-stuck-at-open*, are the same and are calculated as the |defective node| / |total node| $\times$ 100%. We set the defect rates as 2%, 3%, and 4% for showing the impact of defect rates on the diagnosis results. The defects were randomly injected into the SET arrays based on the defect rates and defect distribution assumed in this paper. According to the related research [9], the height constraint of SET arrays limits the number of inputs in an SET array. In [9], the height constraint was suggested as ten. Therefore, we conduct the experiments for SET arrays with different sizes from 10 $\times$ 10 to 60 $\times$ 60.

In the first experiment, for a single run, we randomly generated a defect map based on the assumption, and performed the step of *finding a conducting path* on it. Then, we calculated the number of trials required for finding a conducting path. We conducted 100 runs of experiments and obtained the average result for each size of SET array.

Table I summarizes the results of the first experiment. Columns 1 and 2 list the dimension of the defective SET arrays. Column 3 lists the different defect rates of the SET arrays. Column 4 lists the number of runs that cannot find a conducting path within 200 trials. Column 5 lists the average number of trials for finding a conducting path excluding the failing runs. Columns 6 and 7 list the average CPU time of the experiments excluding that in the failing runs and the total average CPU time. Columns 8–14 are the results as Columns 1–7.

TABLE I
EXPERIMENTAL RESULTS OF THE STEP OF FINDING A CONDUCTING PATH

Height	Width	Defect (%)	|Failing run|	|Trial|	AT (s)	TAT (s)	Height	Width	Defect (%)	|Failing run|	|Trial|	AT (s)	TAT (s)
10	10	2	7	1.38	0.78	9.32	20	40	2	3	2.11	1.08	4.01
		3	2	1.51	1.07	3.43			3	2	2.71	1.67	3.62
		4	6	2.31	3.12	9.12			4	4	4.21	3.18	7.01
	15	2	3	1.28	0.25	3.55		50	2	2	2.18	1.16	3.12
		3	2	1.55	0.56	2.79			3	4	3.18	2.14	6.01
		4	7	2.04	1.10	8.82			4	5	4.01	2.97	7.77
	20	2	3	1.47	0.46	3.72	25	25	2	2	2.56	1.56	3.52
		3	4	1.54	0.53	4.95			3	6	3.45	2.51	8.36
		4	5	1.88	0.97	6.36			4	3	6.81	5.91	8.75
	30	2	1	1.23	0.22	1.26		30	2	3	2.78	1.79	4.72
		3	4	1.77	0.77	5.05			3	6	3.72	2.71	8.51
		4	5	2.21	1.20	6.69			4	3	5.77	4.77	7.61
	40	2	2	1.58	0.57	2.66		40	2	0	2.85	1.82	1.82
		3	2	1.74	0.73	2.87			3	3	4.28	3.26	6.13
		4	5	1.81	0.80	6.14			4	8	5.59	4.57	12.12
	50	2	3	1.16	0.15	3.54		50	2	2	2.37	1.35	3.30
		3	4	1.91	0.89	5.21			3	1	3.77	2.75	3.71
		4	7	2.02	1.00	8.59			4	1	6.62	5.60	6.53
15	15	2	2	1.53	0.60	2.72	30	30	2	1	3.04	2.02	2.99
		3	1	2.07	1.25	2.29			3	0	6.01	5.02	5.02
		4	6	2.37	1.56	7.71			4	4	9.78	8.86	12.49
	20	2	4	1.57	0.58	4.55		40	2	2	2.21	1.19	3.15
		3	1	2.01	1.01	2.01			3	3	4.05	3.03	5.91
		4	4	2.94	1.93	5.90			4	8	8.03	6.99	14.35
	30	2	0	1.47	0.45	0.45		50	2	0	2.88	1.86	1.86
		3	5	2.07	1.05	5.97			3	3	4.24	3.21	6.09
		4	6	2.89	1.86	7.71			4	4	8.32	7.29	10.96
	40	2	5	1.81	0.77	5.68	40	40	2	29	3.46	2.63	31.14
		3	2	2.17	1.16	3.13			3	42	4.57	3.74	44.66
		4	4	4.06	3.04	6.90			4	46	9.04	8.35	51.09
	50	2	1	1.80	0.78	1.76		50	2	23	2.47	1.44	23.93
		3	6	2.33	1.31	7.19			3	35	6.38	5.36	38.26
		4	5	2.67	1.65	6.53			4	58	9.24	8.20	60.96
20	20	2	2	2.20	1.24	3.23	50	50	2	47	2.43	2.44	55.71
		3	6	3.09	2.18	8.16			3	62	4.03	4.51	73.16
		4	7	4.97	4.15	10.94			4	68	10.25	13.94	82.68
	30	2	1	2.14	1.13	2.10	60	60	2	58	3.21	22.24	109.91
		3	4	2.97	1.94	5.83			3	81	6.42	35.86	146.92
		4	3	4.51	3.47	6.34			4	92	10.38	71.35	164.40
Total	-	-	871	273.44	318.61	1143.52							
Average	-	-	11.17	3.51	4.08	14.85							

For example, in the SET array of 20 × 50 with the defect rate of 4%, there were 5 out of 100 runs that cannot find a conducting path under 200 trials. In the remaining 95 runs, on average, 4.01 trials were required to find a conducting path and the average CPU time was 2.97 s. The total average CPU time was 7.77 s. Furthermore, the total average CPU time among all sizes of SET arrays was 14.85 s.

According to Table I, the average number of trials of all the sizes of defective SET arrays for finding a conducting path is less than 4. Therefore, the step of *finding a conducting path* is practical to find a baseline path for the succeeding diagnosis procedure in the proposed algorithm. However, for some random defect maps, a conducting path cannot be found within 200 trials. For these cases, we just discard these SET arrays. In general, the likelihood of having this situation is proportional to the magnitude of a defect rate. Furthermore, when the height of an SET array is greater than or equal to 40, which represents the number of input variables in a circuit, the number of failing runs significantly increases. This is because the probability of a random path that is blocked by open defects is positively correlated with the height of the SET array. This probability can be calculated as $1-(1-open-defect\ rate)^{height}$. However, in the experiments, we still set 200 trials as a threshold to determine if a defective SET array fails or not even the probability of finding a conducting path becomes lower. Note that the static approach in [12] did not suffer from the failing runs, since it always tries all the paths in the defective SET array.

In the second experiment, we also generated defect maps for each size of SET array with different defect rates, and diagnosed them using the algorithms in the state of the art [12] and this paper. We conducted 20 runs of experiments to obtain the average result for each size of SET arrays.

Table II summarizes the experimental results of the second experiment. Columns 1 and 2 list the dimension of the defective SET arrays. Column 3 lists the defect rate of the SET arrays. Column 4 lists the coverage of the diagnosed defects. The coverage is calculated as

$$\text{coverage} = \frac{|\text{d.d.}|}{|\text{t.d.}| - |\text{u.e.d.}| - |\text{u.n.d.}| - |\text{x.d.}|} \times 100\%$$

where |d.d.| denotes the number of diagnosed defects, |t.d.| denotes the number of total defects, |u.e.d.| denotes the number of defects in the undiagnosable edges, |u.n.d.| denotes the

TABLE II
EXPERIMENTAL RESULTS OF THE PROPOSED DYNAMIC DIAGNOSIS APPROACH AND [12]

Height	Width	Defect (%)	Coverage (%)	False-negative (%)	|Config.|	|Pattern|	Average time (s)	Average time of [12] (s)	Ratio (%)
10	10	2	100	0.00	287.4	105.6	1.00	0.00	0.00
		3	100	0.30	287.1	104.2	1.46	0.00	0.00
		4	100	0.40	303.1	108.2	3.54	0.00	0.00
	15	2	100	0.53	449.2	166.0	0.34	0.01	0.03
		3	100	0.80	467.6	173.5	0.57	0.01	0.02
		4	100	0.47	475.6	171.6	1.11	0.01	0.01
	20	2	100	0.95	621.0	228.3	0.47	0.02	0.04
		3	100	1.20	627.1	228.6	0.54	0.02	0.04
		4	100	2.40	639.0	229.6	0.98	0.02	0.02
	30	2	100	1.20	938.2	349.1	0.24	0.04	0.16
		3	100	1.20	968.6	350.7	0.79	0.04	0.05
		4	100	2.67	989.2	350.3	1.23	0.04	0.03
	40	2	100	2.25	1284.6	467.0	0.61	0.07	0.11
		3	100	2.50	1283.9	460.8	0.77	0.07	0.09
		4	100	2.83	1307.3	470.4	0.84	0.06	0.08
	50	2	100	2.38	1560.3	581.1	0.21	0.09	0.41
		3	100	2.46	1641.9	586.5	0.95	0.09	0.09
		4	100	4.00	1660.0	595.7	1.06	0.05	0.05
15	15	2	100	0.84	676.3	254.4	0.73	0.50	0.69
		3	100	1.69	684.7	254.9	1.30	0.49	0.37
		4	100	1.82	693.8	257.0	1.58	0.42	0.26
	20	2	100	0.63	899.5	344.7	0.60	0.92	1.53
		3	100	0.90	911.4	347.8	1.03	0.84	0.81
		4	100	2.20	933.3	348.5	1.95	0.81	0.41
	30	2	100	1.42	1343.0	519.5	0.49	1.95	3.98
		3	100	1.60	1367.4	522.4	1.09	1.72	1.57
		4	100	2.27	1411.6	532.9	1.90	1.71	0.90
	40	2	100	0.90	1842.9	704.0	0.84	2.89	3.46
		3	100	2.70	1900.4	725.1	1.23	2.94	2.40
		4	100	2.40	1976.7	714.3	3.11	2.48	0.80
	50	2	100	1.01	2346.5	880.0	0.88	5.03	5.73
		3	100	2.27	2359.0	886.9	1.41	4.70	3.35
		4	100	2.51	2379.4	901.7	1.75	3.56	2.04
20	20	2	100	0.85	1207.2	453.4	1.28	40.19	31.42
		3	100	1.55	1259.9	467.8	2.21	33.40	15.09
		4	100	3.00	1328.0	472.6	4.19	31.57	7.54
	30	2	100	0.63	1864.0	700.0	1.19	71.45	59.97
		3	100	1.60	1913.4	713.8	2.00	69.88	34.95
		4	100	1.60	1984.6	722.8	3.53	51.2	14.50
	40	2	100	0.63	2487.4	944.5	1.18	139.25	118.06
		3	100	1.05	2551.2	949.4	1.77	119.45	67.52
		4	100	1.70	2653.6	964.3	3.28	89.62	27.35
	50	2	100	0.52	3143.2	1192.2	1.30	218.25	167.76
		3	100	0.94	3242.7	1202.5	2.28	175.98	77.05
		4	100	1.56	3319.9	1209.6	3.11	176.31	56.63
25	25	2	100	0.77	1919.0	730.4	1.62	>3600	-
		3	100	1.15	1950.7	731.8	2.57	>3600	-
		4	100	1.60	2126.3	750.2	5.97	>3600	-
	30	2	100	0.24	2318.8	873.6	1.87	>3600	-
		3	100	1.84	2385.3	887.3	2.79	>3600	-
		4	100	1.57	2482.0	902.7	4.85	>3600	-
	40	2	100	0.68	3129.2	1189.2	1.95	>3600	-
		3	100	1.66	3267.2	1204.4	3.39	>3600	-
		4	100	1.52	3363.0	1211.4	4.70	>3600	-
	50	2	100	0.56	3947.5	1498.5	1.55	>3600	-
		3	100	0.98	4032.7	1530.7	2.95	>3600	-
		4	100	2.51	4363.5	1549.1	5.80	>3600	-
30	30	2	100	0.44	2765.8	1059.7	2.14	>3600	-
		3	100	1.29	2947.9	1080.2	5.15	>3600	-
		4	100	1.62	3196.8	1105.1	8.97	>3600	-
	40	2	100	0.55	3659.3	1415.9	1.37	>3600	-
		3	100	1.30	3794.0	1433.8	3.20	>3600	-
		4	100	1.34	4184.4	1486.8	7.17	>3600	-
	50	2	100	0.77	4694.1	1818.6	2.12	>3600	-
		3	100	1.59	4789.4	1801.9	3.47	>3600	-
		4	100	1.19	5151.7	1798.4	7.55	>3600	-
40	40	2	100	0.74	5083.0	1998.9	2.93	>3600	-
		3	100	1.15	5162.9	2015.5	4.04	>3600	-
		4	100	1.46	5559.6	2054.6	8.76	>3600	-
	50	2	100	0.57	6312.6	2529.1	1.86	>3600	-
		3	100	0.86	6701.6	2555.1	5.77	>3600	-
		4	100	1.36	6978.6	2573.5	8.62	>3600	-
50	50	2	100	0.72	8066.1	3308.6	3.35	>3600	-
		3	100	1.16	8234.0	3334.6	5.62	>3600	-
		4	100	1.46	8790.7	3370.4	15.09	>3600	-
60	60	2	100	0.61	11927.6	4943.15	27.07	>3600	-
		3	100	1.20	12101.5	4967.10	41.69	>3600	-
		4	100	1.31	12668.1	4997.95	771.7	>3600	-
Total	-	-	-	-	232527.0	88620.75	1041.56	-	-
Average	-	-	-	-	2981.1	1136.16	13.35	-	-

number of defects in the useless nodes, and |x.d.| denotes the number of don't-care defects. Column 5 lists the false negative, which is the percentage that defect-free edges were considered as defective ones, and is calculated as |misjudged edge| / |total edge| $\times$ 100%. Columns 6 and 7 list the average numbers of node configurations and input patterns that are used in the proposed diagnosis approach for each SET array. Columns 8 and 9 list the average CPU time of our approach and [12].

TABLE III
EXPERIMENTAL RESULTS OF NUMBER OF UNDIAGNOSABLE EDGES AND USELESS NODES, AND THE NUMBER OF DEFECTS IN THEM

Height	Width	Defect (%)	|u.e + u.n|	|Defect| in u.e + u.n	Height	Width	Defect (%)	|u.e + u.n|	|Defect| in u.e + u.n
10	10	2	22	0	20	40	2	45.75	6.25
		3	22	0.25			3	44.75	6.5
		4	22	0.25			4	50.75	13.75
	15	2	21.25	0.75		50	2	41.25	1.75
		3	21.75	1.25			3	49.5	11.25
		4	21	0.25			4	51.25	14.25
	20	2	22.25	0.75	25	25	2	53.75	5.75
		3	23.25	2			3	53.75	7.25
		4	26	5			4	55.5	10.5
	30	2	23	1.5		30	2	51	2
		3	24.5	4.5			3	59.5	12.25
		4	32.25	13			4	58	12.25
	40	2	24.75	4		40	2	53.5	4.5
		3	36.5	16.5			3	63.5	16.25
		4	34.5	15.25			4	60	14.25
	50	2	34.25	13.5		50	2	55.5	6.5
		3	33.75	13.75			3	62	14.75
		4	34.5	15.25			4	79	33.25
15	15	2	31.5	2.25	30	30	2	63.25	5
		3	30.75	2.5			3	65.25	8.75
		4	31	3.75			4	71.5	17
	20	2	32.75	2.75		40	2	62	3.75
		3	31.5	2.5			3	70.5	14
		4	35	6.75			4	73.75	19.25
	30	2	34	4		50	2	72.5	14.25
		3	35.25	6.25			3	79.5	23
		4	36	7.75			4	70	15.5
	40	2	34.75	4.75	40	40	2	88.25	11.25
		3	46	17			3	93.75	19.25
		4	46	17.75			4	97.75	25.5
	50	2	37.5	7.5		50	2	86.75	9.75
		3	47	18			3	92.5	18
		4	46.75	18.5			4	95.25	23
20	20	2	43.5	4	50	50	2	115.25	19.25
		3	44.75	6.5			3	122	29.25
		4	51	14			4	132	42.25
	30	2	43.5	4	60	60	2	140.25	25.5
		3	47	8.75			3	145.25	34.25
		4	46.75	9.75			4	159.25	52
Total	-	-	4296.5	899.75					
Average	-	-	108.77	22.78					

The last column lists the ratio of CPU time between [12] and our approach. The last row shows the average number of node configurations and input patterns, and the average CPU time for all the SET arrays.

For example, in the SET array of 20 × 50 with the defect rate of 4% for each defect type, the false negative was 1.56%, and 3319.9 configurations and 1209.6 patterns were required on average. The average CPU time of this paper and [12] was 3.11 and 176.31 s, respectively. The ratio of CPU time between the two approaches was 56.63. The last row shows that the average CPU time for all the SET arrays in the proposed approach was 13.35 s.

According to Table II, the proposed diagnosis approach can achieve 100% coverage, which is the same as [12]. However, for the SET arrays with height larger than or equal to 25, the CPU time of [12] exceeded the CPU time limit, 3600 s, of this paper, while our approach only cost a few seconds. Thus, the proposed dynamic approach is more efficient and scalable.

For some cases, the percentage of false negative was large. This is the situation when most of the defects did not occur in the undiagnosable edges and useless nodes. Under this situation, most undiagnosable edges and useless nodes are defect-free, but they are misjudged as open defect in our approach. Therefore, the percentage of false negative would be large.

Nevertheless, since these false-negative edges cannot pass electrons in any situation and have the same defect effect as open-defect edges, we mark them as having open defects in our approach. Besides, since the defect-reuse technique in the defect-aware synthesis algorithm [12] only reuses the edges with short defects, not reuses the edges with open defects for path configurations, the false-negative edges will never be used, such that no erroneous mapping results will occur.

In the third experiment, we also generated defect maps for each size of SET array with different defect rates. We conducted four runs of experiments to obtain the number of undiagnosable edges and useless nodes in the SET array, and the number of defects in them. Table III summarizes the experimental results of the third experiment. Column 4 lists the average number of undiagnosable edges and useless nodes. Column 5 lists the average number of defects in them. Columns 6–10 are the results as Columns 1–5.

For example, in the SET array of 20×50 with the defect rate of 4%, the average number of undiagnosable edges and useless nodes is 51.25, and the average number of defects in them is 14.25. Furthermore, according to Table III, the average number of undiagnosable edges and useless nodes among all sizes of SET arrays is 108.77, and the average number of defects in them among all sizes of SET arrays is 22.78.

V. Conclusion

The presence of defects is very common in nanotechnology. It is also the case for the SET devices. To elevate the reliability of SET arrays, we propose the first dynamic diagnosis approach for defective SET arrays. The major difference between the previous work and the proposed dynamic diagnosis approach is that the dynamic approach adjusts the diagnosis sequence by the feedbacks of the previous diagnosis process. Thus, the required configurations and input patterns can be reduced. The experimental result also shows that the proposed diagnosis approach can achieve 100% coverage as the state of the art, but with much less CPU time. With the dynamic diagnosis approach, the synthesis flow of defective reconfigurable SET arrays will become more efficient and complete.

References

[1] N. Asahi, M. Akazawa, and Y. Amemiya, "Single-electron logic device based on the binary decision diagram," *IEEE Trans. Electron Devices*, vol. 44, no. 7, pp. 1109–1116, Jul. 1997.

[2] 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.*, Jun. 2011, pp. 878–883.

[3] 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. Emerg. Technol. Comput. Syst.*, vol. 9, no. 1, Feb. 2013, Art. no. 5.

[4] Y.-H. Chen, J.-Y. Chen, and J.-D. Huang, "Area minimization synthesis for reconfigurable single-electron transistor arrays with fabrication constraints," in *Proc. Design, Autom. Test Eur.*, Mar. 2014, pp. 1–4.

[5] Y.-H. Chen, Y. Chen, and J.-D. Huang, "ROBDD-based area minimization synthesis for reconfigurable single-electron transistor arrays," in *Proc. Int. Symp. VLSI Design, Autom. Test*, Mar. 2015, pp. 1–4.

[6] C.-E. Chiang *et al.*, "On reconfigurable single-electron transistor arrays synthesis using reordering techniques," in *Proc. Conf. Design, Autom. Test Eur.*, Mar. 2013, pp. 1807–1812.

[7] S. Eachempati, V. Saripalli, N. Vijaykrishnan, and S. Datta, "Reconfigurable BDD-based quantum circuits," in *Proc. Int. Symp. Nanosc. Archit.*, Jun. 2008, pp. 61–67.

[8] 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," *Phys. E, Low-Dimensional Syst. Nanostruct.*, vol. 11, nos. 2–3, pp. 149–154, Oct. 2001.

[9] C.-H. Ho *et al.*, "Area-aware decomposition for single-electron transistor arrays," *ACM Trans. Design Autom. Electron. Syst.*, vol. 21, no. 4, Sep. 2016, Art. no. 70.

[10] S.-Y. Huang and K.-T. Cheng, "ErrorTracer: A fault simulation based approach to design error diagnosis," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 18, no. 9, pp. 1341–1352, Sep. 1999.

[11] C.-Y. Huang, C.-W. Liu, C.-Y. Wang, Y.-C. Chen, S. Datta, and V. Narayanan, "A defect-aware approach for mapping reconfigurable single-electron transistor arrays," in *Proc. Asia South Pacific Design Autom. Conf.*, Jan. 2015, pp. 118–123.

[12] C.-Y. Huang *et al.*, "Diagnosis and synthesis for defective reconfigurable single-electron transistor arrays," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 24, no. 6, pp. 2321–2334, Jun. 2016.

[13] 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.*, Dec. 2001, pp. 622–625.

[14] W. H. Kautz, "Fault testing and diagnosis in combinational digital circuits," *IEEE Trans. Comput.*, vol. C-17, no. 4, pp. 352–366, Apr. 1968.

[15] L. Heh-Tyan, T. Jia-Horng, and L. Chen-Shang, "Efficient automatic diagnosis of digital circuits," in *Proc. IEEE Int. Conf. Comput.-Aided Design*, Nov. 1990, pp. 464–467.

[16] C.-W. Liu *et al.*, "Width minimization in the single-electron transistor array synthesis," in *Proc. Design, Autom. Test Eur.*, Mar. 2014, pp. 1–4.

[17] C.-W. Liu *et al.*, "Synthesis for width minimization in the single-electron transistor array," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 23, no. 12, pp. 2862–2875, Dec. 2015.

[18] L. Liu, X. Li, V. Narayanan, and S. Datta, "A reconfigurable low-power BDD logic architecture using ferroelectric single-electron transistors," *IEEE Trans. Electron Devices*, vol. 62, no. 3, pp. 1052–1057, Mar. 2015.

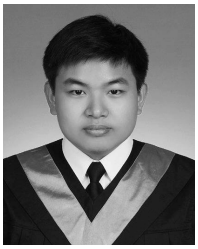
[19] H. W. C. Postma, T. Teepen, Z. Yao, M. Grifoni, and C. Dekker, "Carbon nanotube single-electron transistors at room temperature," *Science*, vol. 293, no. 5527, pp. 76–79, Jul. 2001.

[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, no. 1, pp. 633–637, 2003.

[21] A. Veneris and I. N. Hajj, "Design error diagnosis and correction via test vector simulation," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 18, no. 12, pp. 1803–1816, Dec. 1999.

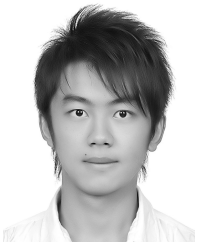
[22] Z. Zhao, C.-W. Liu, C.-Y. Wang, and W. Qian, "BDD-based synthesis of reconfigurable single-electron transistor array," in *Proc. Int. Conf. Comput.-Aided Design*, Nov. 2014, pp. 47–54.

[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, 1998.



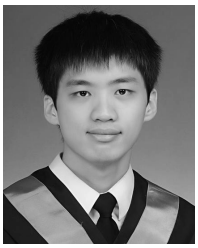
Yun-Jui Li received the B.S. degree from the Department of Computer Science, National Tsing Hua University, Hsinchu, Taiwan, in 2014, where he is currently pursuing the M.S. degree with the Department of Computer Science.

His current research interests include diagnosis and logic synthesis for emerging technologies.



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 with the Department of Computer Science.

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



Chia-Cheng Wu received the B.S. degree from the Department of Computer Science, National Tsing Hua University, Hsinchu, Taiwan, in 2015, where he is currently pursuing the M.S. degree with the Department of Computer Science.

His current research interests include diagnosis and logic synthesis for emerging technologies.

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 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 (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 a Professor. He has authored or co-authored over 50 technical papers in his research fields and holds eight 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 research results were nominated as the Best Papers at the 2009 IEEE Asia and South Pacific Design Automation Conference and the 2010 IEEE/ACM Design Automation Conference, respectively.

Suman Datta (F'13) received the B.S. degree in electrical engineering from IIT Kanpur, Kanpur, India, in 1995, and the Ph.D. degree in electrical and computer engineering from the University of Cincinnati, Cincinnati, OH, USA, in 1999.

He was instrumental in the demonstration of indium antimonide-based quantum-well transistors operating at room temperature with a record energy delay product, the first experimental demonstration of metal gate plasmon screening and channel strain engineering in high-/metal gate CMOS transistors, and the investigation of the transport properties in nonplanar Trigate Transistors. He is currently a Chang Family Chair Professor with the College of Engineering, University of Notre Dame, Notre Dame, IN, USA. He holds over 160 U.S. patents. His current research interests include exploring new materials and novel device architecture for CMOS enhancement and replacement for future energy-efficient computing applications.

Dr. Datta was a member with the Logic Technology Development Group, Intel Corporation, Santa Clara, CA, USA, from 1999 to 2007. He is also a Distinguished Lecturer of the IEEE Electron Devices Society.

Vijaykrishnan Narayanan (F'11) received the B.S. degree in computer science and engineering from the University of Madras, Chennai, India, in 1993, and the Ph.D. degree in computer science and engineering from the University of South Florida, Tampa, FL, USA, in 1998.

He is currently a Professor of Computer Science and Engineering and Electrical Engineering with Pennsylvania State University, University Park, PA, USA. His current research interests include power-aware and reliable systems, embedded systems, nanoscale devices and interactions with system architectures, reconfigurable systems, computer architectures, network-on-chips, and domain specific computing.

Dr. Narayanan was a recipient of several awards, including the Penn State Engineering Society Outstanding Research Award in 2006, the IEEE CAS VLSI TRANSACTIONS Best Paper Award in 2002, the Penn State CSE Faculty Teaching Award in 2002, the ACM SIGDA Outstanding New Faculty Award in 2000, the Upsilon Pi Epsilon Award for Academic Excellence in 1997, the IEEE Computer Society Richard E. Merwin Award in 1996, and the University of Madras First Rank in Computer Science and Engineering in 1993. He has received several certificates of appreciation for outstanding service from ACM and the IEEE Computer Society. He is currently an Editor-in- Chief of the IEEE TRANSACTIONS ON COMPUTER AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS.