

# Diagnosis and Synthesis for Defective Reconfigurable Single-Electron Transistor Arrays

Ching-Yi Huang, Yun-Jui Li, Chian-Wei Liu, Chun-Yao Wang, *Member, IEEE*, Yung-Chih Chen, 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. However, early realizations of SET array lacked variability and reliability due to their fixed architectures and high defect rates of nanowire segments. Therefore, a reconfigurable version of SET was proposed to deal with these issues. Recently, several automated mapping approaches have been proposed for area minimization of reconfigurable SET arrays. However, to the best of our knowledge, seldom mapping algorithms that consider the existence of defective nanowire segments were proposed. Furthermore, before the defect-aware mapping, we have to know the locations of defects in SET arrays. Thus, this paper presents the first diagnosis approach to identify the locations of defects in SET arrays followed by two defect-aware algorithms for mapping SET arrays in different scenarios. The experimental results show that the proposed diagnosis method can detect 100% of defects under a defect rate and distribution in SET arrays. As for the mapping algorithms, the results show that our approach can successfully map the SET arrays with 11.13% and 7.69% width overhead on average in the baseline detour mapping algorithm and defect-reuse mapping algorithm, respectively, in the presence of 5000-ppm defects.

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

## I. INTRODUCTION

**P**OWER consumption has become one of the primary bottlenecks to meet Moore’s law. To deal with this issue, many emerging low-power devices have been

Manuscript received July 22, 2015; revised October 19, 2015; accepted November 25, 2015. Date of publication January 6, 2016; date of current version May 20, 2016. This work was supported by the Ministry of Science and Technology, 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.

C.-Y. Huang, Y.-J. Li, C.-W. Liu, and C.-Y. Wang are with the Department of Computer Science, National Tsing Hua University, Hsinchu 30013, Taiwan (e-mail: s986516@m98.nthu.edu.tw; abc61219@yahoo.com.tw; smilelcw@gmail.com; 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 Department of Electrical Engineering, Pennsylvania State University, University Park, PA 16802 USA (e-mail: sdatta@engr.psu.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.2015.2506780

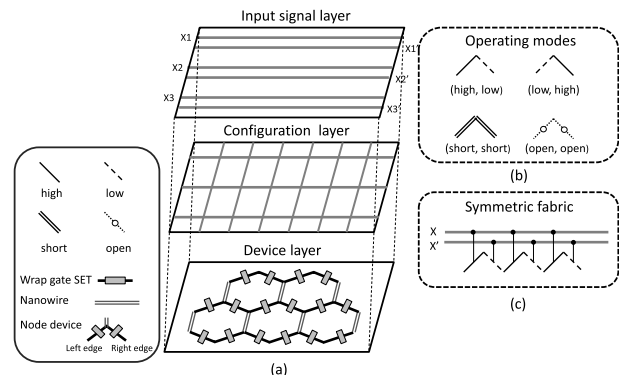


Fig. 1. (a) Physical architecture of a reconfigurable SET array [12]. (b) Operating modes. (c) Symmetric fabric architecture for input wiring.

explored recently. Among these devices, some demonstrations of operations of single-electron transistors (SETs) at room temperature have proved that these devices are promising candidates to substitute traditional CMOS devices for future designs [1]–[4].

Since only a few electrons are involved in the switching process, SETs suffer from low transconductance. Therefore, the conduction mechanism of the CMOS-based logic is not applicable to SETs. To this end, a binary decision diagram (BDD)-based [5] architecture was proposed as a platform for implementing logic functions using SETs [6]. Using this, a Boolean circuit can be implemented through mapping the BDD of the Boolean function onto an SET array [7], [8]. However, the realization of the architecture proposed in [8] is fixed and not amendable to functional reconfiguration. Furthermore, if any of the nanowire segments or the SETs is defective, the whole circuit becomes useless. This causes its low yield due to a high defect rate of nanowires and nanodevices. Fortunately, a reconfigurable version of SET using wrap gate tunable tunnel barriers was proposed [9] to increase the flexibility and reliability of the SET arrays [6]–[8]. The electrostatic properties through in-depth device simulation were also presented in [10] and [11], which showed that this device can provide an energy–delay product that is an order of magnitude lower than traditional CMOS devices.

Fig. 1(a) illustrates the structure of the reconfigurable SET array adapted from [12], which can be divided into three layers. The bottom layer is the device layer, which is composed of the hexagonal nanowire network with wrap gate SETs.

The middle layer is used to configure the operating modes of every transistor. The top layer takes charge of input signal connections to SETs. If some nanowires or SETs (bottom layer) are defective after manufacturing, we still have a usable SET array through configuration in the middle layer.

Recently, this success of SET array realization has attracted the development of its automation tools [12]–[19]. The first automated synthesis approach was proposed in [13], which presented a *product-term-based mapping approach* that synthesizes a logic circuit by mapping all its product terms into the SET architecture. Based on the product-term-based approach, Chiang *et al.* [15] proposed mapping approaches to reduce the number of hexagons of the mapped SET arrays under *symmetric fabric constraint using product-term reordering and variable reordering techniques*. However, the area of an SET array on a chip is the product of its bounded height and width. As a result, Chen *et al.* [12], [16] and Liu *et al.* [17], [18] proposed width minimization approaches. The algorithm proposed in [12] dynamically chose the best unmapped product terms to achieve a maximal sharing. References [16], [17] and [18] focus on minimizing the number of product terms to obtain more compact SET arrays.

In contrast to the product-term-based mapping approaches, Zhao *et al.* [19] proposed a *BDD-based mapping approach*, which exploited the structure similarity between an SET array and a BDD, to synthesize SET arrays.

Although these previous works are effective, they all assume that the device layer of the SET array is defect free. This assumption, however, does not match the real situation. Therefore, if some nanowires or SET devices are defective after manufacturing, the previous approaches might fail to map. This is because the mapping process must avoid the defective areas for correct mapping.

To consider the reliability issue of SET arrays and establish a robust automation flow, two important techniques must be developed: 1) a diagnosis method for identifying the locations of defects in an SET array and 2) a defect-aware mapping approach for mapping a Boolean circuit into a defective SET array with respect to the locations of defects detected by the diagnosis process. Although Huang *et al.* [20] have proposed the first defect-aware approach for mapping SET arrays, some details of the algorithms were not introduced and considered. Besides, the algorithm was not well integrated with a diagnosis process into a complete defect-aware mapping flow. As a result, in this paper, we propose a diagnosis approach for SET arrays and a defect-aware mapping flow for mapping defective SET arrays.

In this paper, we first propose a defect model, considering three types of defects: *single-stuck-at-open*, *double-stuck-at-open*, and *stuck-at-short*, on SET arrays. We also propose an assumption about the defect distribution in the array. To diagnose an SET array, we propose an *expand-and-all-pass* traversing method, which can achieve a 100% defect coverage under the assumption of defect distribution. In the defect-aware mapping algorithm, we propose a *baseline detour mapping algorithm* and a *defect-reuse mapping algorithm*.

We divide the experiments into two parts in this paper. In the first part, the experiment was conducted for verifying

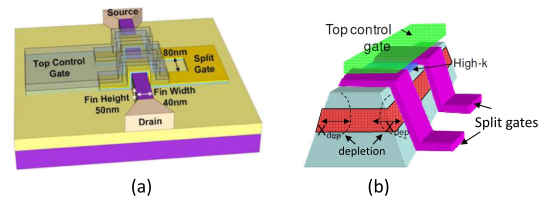


Fig. 2. (a) Structure of a reconfigurable SET device [21]. (b) Formulations of wrap-around Schottky split gates and the top control gate [9].

the validity of the diagnosis method. In the second part, we conducted two sets of experiments for the defect-aware mapping algorithms over a set of IWLS2005 benchmarks [24]. In the first set of experiment, we uniformly and randomly injected 0.5% or 5000-ppm defects into SET arrays. This defect rate is a user-defined parameter, and 0.5% is higher than that in the conventional MOS process, which reflects the vulnerability of nanowires and nanodevices. The experimental results show that our approach can successfully map the SET arrays. The width increases are 11.13% and 7.69% on average in the baseline detour algorithm and defect-reuse algorithm, respectively, compared with the defect-free SET arrays.

In the second set of experiments, we used different defect rates and observed the corresponding width variation for some benchmarks. The experimental results show that the width is increased slowly in the defect-reuse algorithm compared with that in the baseline detour algorithm.

Finally, we also provide our diagnosis and mapping tool with graphical user interface (GUI) to present the mapping results of SET array. The link for downloading the tool is released at [25].

## II. PRELIMINARIES

### A. Reconfigurable SET

As shown in Fig. 1(a), the bottom layer of an SET array is composed of the hexagonal nanowire network with reconfigurable SET devices. The structure of a reconfigurable SET device is shown in Fig. 2 [9], [21], where a pair of Schottky gates, called split gates, are wrapped around the fin that connects the source and drain, and the top control gate is built upon the splits gates. Specifically, the split gates are connected to the configuration grid in the middle layer of the SET array, and the top control gate is connected to the input signals in the top layer.

By providing the split gates a voltage bias through the grid in the middle layer, 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

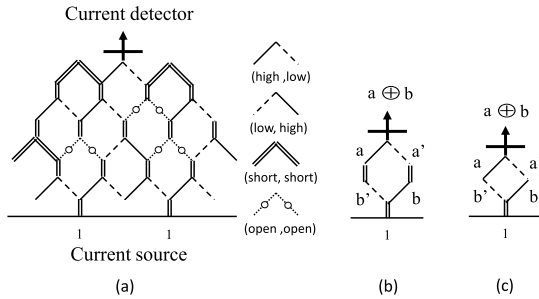


Fig. 3. (a) SET array. (b) Example of  $a \oplus b$ . (c) Simplified diamond-shaped network of  $a \oplus b$  [13].

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.

### B. Reconfigurable SET Array

A reconfigurable SET array can be represented as a hexagonal network as shown in Fig. 3(a). At the top of the SET array, there is a current detector measuring the current coming from the current source, represented as 1, at the bottom. When the electrons transported from the current sources are detected through a conducting path, which is controlled by the input variables, the output value is 1; otherwise, it is 0.

Each sloping edge in the SET array represents an SET device and can be configured as one of four modes: *high*, *low*, *short*, or *open*. A *high* (*low*) edge indicates that the corresponding SET device operates in the active mode controlled by a variable  $x$  ( $x'$ ). Furthermore, a *short* (*open*) edge is electrical *short* (*open*), where the corresponding SET device operates in the *short* (*open*) mode. The connections to the current source can also be configured as either *short* for connection or *open* for disconnection. For example, Fig. 3(b) shows an implementation of  $a \oplus b$ . Since all the vertical edges of the hexagons are electrically *short*, for the ease of discussion, only the sloping edges are preserved in the abstract graph. Fig. 3(c) shows the corresponding diamond-shaped abstract network of the hexagonal network in Fig. 3(b).

### C. Symmetric Fabric Constraint

To reduce the wiring area of SET arrays, *symmetric fabric constraint* [9] was imposed in all the previous works. The symmetric fabric constraint enforces that a pair of left and right edges of a *node device* must be one of (*high*, *low*), (*low*, *high*), (*short*, *short*), or (*open*, *open*), as shown in Fig. 1(b). Moreover, the constraint enforces that both (*high*, *low*) and (*low*, *high*) configurations are not allowed to appear in the same row simultaneously, as shown in Fig. 1(c).

Since the determination of the configuration architecture, either (*high*, *low*) or (*low*, *high*) for one row, is independent of the proposed diagnosis methodology, the configuration architecture of each row is assumed to be (*low*, *high*) for the ease of discussion in Section IV.

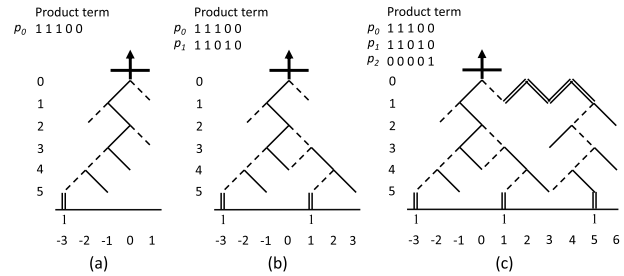


Fig. 4. Example of product-term-based mapping procedure. (a)  $p_0$ . (b)  $p_0 + p_1$ . (c)  $p_0 + p_1 + p_2$ .

### D. Product-Term-Based Mapping Approach

The objective of the product-term-based mapping approach is to configure a path in the SET array for each product term while avoid creating invalid paths that correspond to invalid product terms. Fig. 4 illustrates an example of the product-term-based mapping procedure. Given the first product term  $p_0 = 11100$ , we start from the root node below the current detector, and find or configure an edge for each bit in  $p_0$  from the top row to the bottom row (also called from a higher level to a lower level). The mapping rule is to configure *high* for 1, *low* for 0, and *short* for don't care, denoted as  $-$ . The mapping result of  $p_0$  is shown in Fig. 4(a). For  $p_1$ , we still map from the root node. Since the first two bits of  $p_1$  are the same as that of  $p_0$ , the configurations at the coordinations of  $(x, y) = (0, 0)$  and  $(-1, 1)$  are shared between these two product terms. Then the mapping path of  $p_1$  is branched from  $(0, 2)$  toward  $(1, 3)$  and then  $(2, 4)$ , as shown in Fig. 4(b). For  $p_2$ , since no configuration can be reused in the mapping result of Fig. 4(b), we use the *expansion* operation with *short* edges at  $(x, y) = (2, 0)$  and  $(4, 0)$  for successfully mapping  $p_2$ , as shown in Fig. 4(c).

### E. Branch-Then-Share

*Branch-then-share* (BTS) product terms are two product terms that branch in one row and merge in the succeeding row such that the remaining edges are all shared [18]. The BTS forms structures that shrink the width of mapping area. In this paper, the configurations for two consecutive rows are classified into two types: 1) twin type and 2) invert type. The twin type (invert type) represents the BTS that occurs at two consecutive rows having the same (opposite) configurations. Please refer to [18] for the details.

Furthermore, we observe that some BTS may form a *BTS group* where these BTS are mapped at consecutive and neighboring locations. Fig. 5(a) shows a mapping result of a BTS group. Fig. 5(b) shows the same example with a different mapping result, where the original BTS group is separated into two BTS groups.

### F. Expansion Operation

As mentioned in Section II-D, the expansion operation is applied when the space for mapping a product term is not enough. In [13], the expansion operation was allowed only at the first row of SET array. Reference [18] then

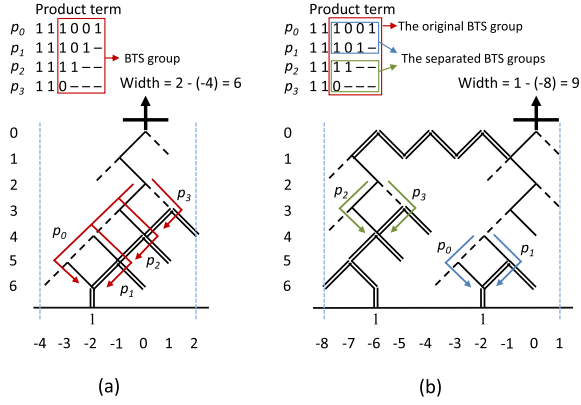


Fig. 5. (a) Example of consecutive BTS for product terms  $p_0 + p_1 + p_2 + p_3$ . (b) Mapping result of  $p_0 + p_1 + p_2 + p_3$  if we do not group these product terms into a consecutive BTS group.

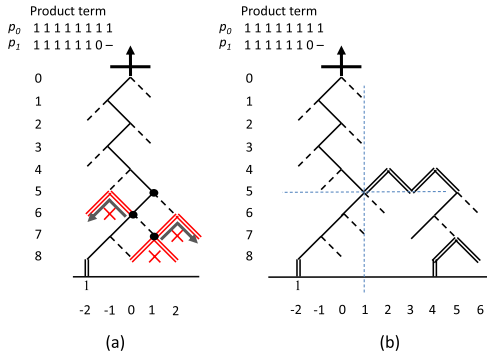


Fig. 6. Example of expansion at any row. (a) Mapping result of  $p_0$ . (b) Mapping results of  $p_0 + p_1$  after a successful expansion.

allows *expansion at any row* for having smaller mapping area. As a result, these algorithms found the lowest feasible location for expansion without causing invalid paths. For example, as shown in Fig. 6(a), to map the product term  $p_1$  after  $p_0$ , the configuration (*short, short*) for the last bit  $-$  at  $(x, y) = (1, 7)$  is not allowed. This is because this configuration will create an invalid conducting path with respect to a nonexistent product term  $11111110$  no matter the current source is located at either  $(0, 8)$  or  $(2, 8)$  for  $p_1$ . Thus, we need to use an expansion operation to map  $p_1$ . In this example,  $(1, 7)$ ,  $(0, 6)$ , and  $(1, 5)$  could be the locations for expansion. However, the expansion operation from  $(1, 7)$  and  $(0, 6)$  will create invalid paths as well. Thus,  $(1, 5)$  is the lowest feasible location for expansion, and the mapping result of  $p_0 + p_1$  is shown in Fig. 6(b).

Furthermore, when considering the BTS group, the lowest feasible location is not always the best choice for expansion. For example, as shown in Fig. 7(a), the lowest expansion location for  $p_2$  is  $(x, y) = (2, 6)$ . However,  $p_2, p_3$ , and  $p_4$  are a BTS group as highlighted. If we expand at  $(x, y) = (2, 6)$  for  $p_2$ , we have to expand again at an upper row for mapping  $p_3$  and  $p_4$ , as shown in Fig. 7(b). In contrast, if the expansion location for  $p_2$  is higher than the branch location of the BTS group, e.g.,  $(1, 3)$ , as shown in Fig. 7(c), the structure with respect to the BTS group can be preserved. As a result, the width of the mapping result is reduced as shown in Fig. 7(d).

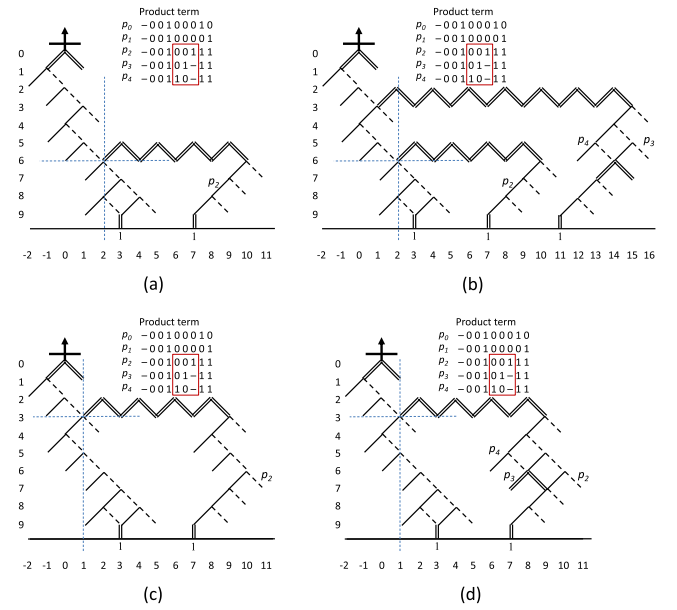


Fig. 7. (a) Example of expansion at the lowest feasible location for  $p_2$ . (b) Mapping result of  $p_0 + p_1 + p_2 + p_3 + p_4$  without considering the BTS group. (c) Example of expansion for  $p_2$  with considering the BTS group of  $p_2 + p_3 + p_4$ . (d) Final mapping result of  $p_0 + p_1 + p_2 + p_3 + p_4$ .

### III. DEFECT MODEL AND DEFECT DISTRIBUTION

In this section, we first introduce the considered defect model, and then propose an assumption about the defect distribution in the SET arrays.

#### A. Defect Model

*Open* and *short* are common types of defects that occur in the SET array. These defects occur when the SET device is defective and results in either a permanent short or open condition based on the operation mechanism of the SET device as mentioned in Section II-A [9]. In addition, the nanowires connected to the SET devices would also suffer from open and short defects caused by common manufacturing defects. Open defect could arise from excessive thinning of the etched fin, which would lead to physical disruption in the continuity of the tunnel junctions in the fin. Open could also arise from over etching of nanowires. On the other hand, short defect could arise from the variation in fin thickness of an SET, which would lead to the variation of the effect from the voltage bias of the split gate on the mode control of the SET. That is, in the active and open modes, the varied effect of the negative voltage bias of the split gate would fail to form a complete depletion region to pinch off tunneling, and results in a permanent short mode. Short could also arise from wire bridging between source and drain.

Fig. 8(a) shows a defect-free node device. If an open defect occurs on an SET device or occurs on a nanowire connected to the SET device, i.e., on the left edge or the right edge of a node device as shown in Fig. 8(b), the electrons transported from the lower node device will never pass through the defective SET device. This single defective edge is named as a *single-stuck-at-open* defect. If an open defect occurs on a vertical

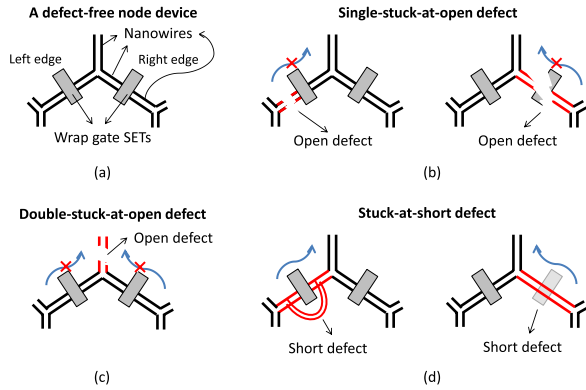


Fig. 8. Proposed defect model. (a) Defect-free node device. (b) Single-stuck-at-open defects. (c) Double-stuck-at-open defect. (d) Stuck-at-short defects.


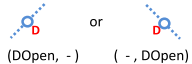




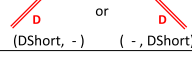

Defect Types	Representation
Single-stuck-at-open: 	 or  (DOpen, -) or (-, DOpen)
Double-stuck-at-open: 	 (DOpen, DOpen)
Stuck-at-short: 	 or  (DShort, -) or (-, DShort)

Fig. 9. Fabric representations of the three types of failures.

nanowire of a hexagon, as shown in Fig. 8(c), the electrons transported from the lower SETs will never pass through the vertical edge, either. In this case, we say that this node device has a *double-stuck-at-open* defect. If a short defect occurs on an SET device or occurs on the nanowires connected to an SET device, as shown in Fig. 8(d), this defective edge will always be conducted. We name this defect a *stuck-at-short* defect. Note that the connections to the current sources could be defective and are also considered in this paper.

In the succeeding discussion, these three types of defects are considered. Their corresponding representations on the network are shown in Fig. 9, where *DOpen* and *DShort* represent the permanent configurations caused by the defects and their functionalities are the same as *open* and *short* configurations, respectively. A node having one of the defects is called a *defective node* in this paper.

### B. Assumption of Defect Distribution

A typical defect rate is usually less than  $0.5\% = 5000$  ppm in the conventional MOS process. Since the SET is more vulnerable than MOS, its defect rate could be as higher as  $2\% = 20000$  ppm. That means two defects occur among 100 node devices on average, and the number of defective nodes is not many in an SET array. Thus, we make two assumptions about the defect distribution in the proposed diagnosis method as follows.

- 1) Both open and short defects do not occur on a defective node simultaneously.
- 2) Two defective nodes are not adjacent to each other.

For example, in Fig. 10(a), if the node device in the middle of the SET array is defective, its six adjacent nodes are assumed

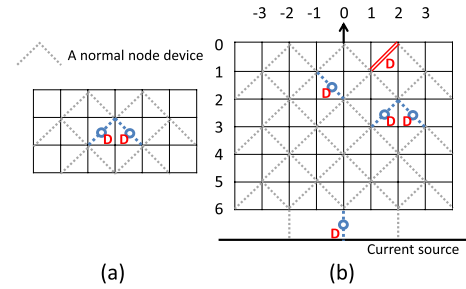


Fig. 10. (a) Adjacency between nodes. (b) Possible defect distribution.

to be normal nodes. Fig. 10(b) shows a possible distribution of defects in an SET array. We can see that three defective nodes are not adjacent. Note that the connections to the current source can be defective as well. For example, an open defect occurs at (0, 6) of Fig. 10(b).

## IV. DIAGNOSIS APPROACH

### A. Overview

Unlike the diagnosis procedure for traditional Boolean circuits [22], [23], we have to configure at least one path between the current detector and the current source of an SET array first before diagnosis. Then the test patterns are generated and cooperated with the corresponding configurations. The problem formulation is as follows.

*Problem Formulation:* Given an  $n$  (height)  $\times$   $m$  (width) SET array with a distribution of defects, we generate the test patterns and determine the specific configurations on node devices. Then we report the locations and the types of the detected defects if exist.

Note that we adopt the static diagnosis process in this paper, which means that the test patterns and the corresponding configurations are predetermined before starting the diagnosis process. Furthermore, any locations of the identified defects will not feedback to the diagnosis program immediately during the diagnosis process. In other words, the test patterns, configurations, and their sequences are not dynamically adjusted. Once the test patterns and the corresponding configurations are generated, all of them must be applied to the array in the diagnosis process sequentially.

We utilize two ideas to identify the defects.

- 1) *If a path between the current detector and the current source is conducted (the SET array outputs 1) under an input pattern, every edge on the path does not suffer from an open defect.*
- 2) *Given a conducted path under an input pattern, if the path is still conducted after reconfiguring a node as (open, open) or reconfiguring the connection to the current source as (open), there is a short defect occurring at the edge or the connection accordingly.*

In other words, after configuring a path and applying a corresponding input pattern, those edges that have no open defects, called *nonopen-defect edges*, can be first identified along the path. Then after reconfiguring each node along the path as *open* mode, we can know whether the nonopen-defect edges are *short-defect edges* or not. For example, the

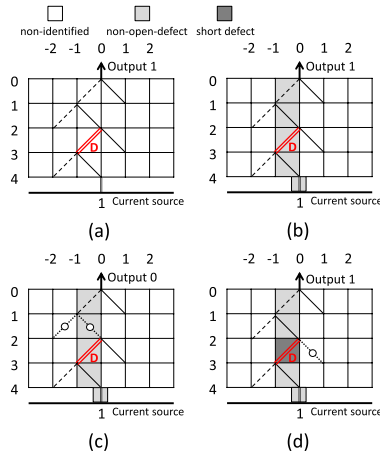


Fig. 11. Example showing the idea of defect identification. (a) Path conducted after applying the input pattern 0101. (b) Identify and mark the nonopen-defect edges along the path. (c) Path is not conducted when the node at  $(-1, 1)$  is reconfigured as  $(open, open)$ . (d) Identify and mark a short defect at  $(0, 2)$  through the  $(open, open)$  reconfiguration at  $(0, 2)$ .

SET array in Fig. 11(a) outputs 1 after applying the input pattern 0101. Thus, we can realize that the edges along the path are nonopen-defect edges, as highlighted in gray boxes in Fig. 11(b). When we reconfigure  $(-1, 1)$  as  $(open, open)$  as shown in Fig. 11(c) and apply the pattern 0101, the output of the SET array becomes 0 (nonconducted), which means that the right edge of the node at  $(-1, 1)$  does not suffer a short defect. However, when we reconfigure  $(0, 2)$  as  $(open, open)$  as shown in Fig. 11(d) and still apply the pattern 0101, the SET array outputs 1 rather than 0. As a result, we can identify that the left edge of the node at  $(0, 2)$  suffers from a short defect, as highlighted in the dark gray box.

To identify *open-defect edges*, on the other hand, we can first identify all the nonopen-defect edges and short-defect edges in an SET array, then the remaining edges, i.e., the edges distinguished from nonopen-defect edges and short-defect edges, are the edges having open defects.

To identify all the defects, we have to configure all the possible paths in the SET array. In this paper, we propose an *expand-and-all-pass* method to diagnose the SET array. Fig. 12 shows the concept of the expand-and-all-pass method. The sequence of the diagnosis process starts from the triangle-shaped *subarray* rooted at  $(0, 0)$ , as shown in Fig. 12(a). Then the diagnosis area, i.e., the subarrays under diagnosis, will be gradually expanded from  $(0, 0)$  to both sides through the expansion operation at the first row, as shown in Fig. 12(b). For each subarray under diagnosis, we configure every possible path within the subarray and apply corresponding patterns to identify nonopen-defect edges and short-defect edges path by path. After checking all the paths in the subarrays, those edges that are not nonopen-defect edges and not short-defect edges are identified as open-defect edges. Finally, the nonopen-defect edges are regarded as normal edges.

However, there are some exceptions that have to be dealt with. First, an open defect occurring at the first row of the SET array can be identified after checking all the paths of two adjacent subarrays (the details will be explained

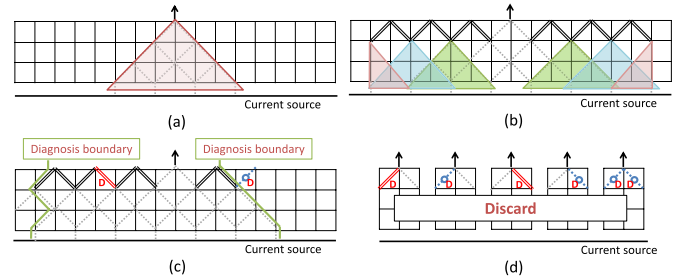


Fig. 12. (a) Abstract of expand-and-all-pass traversing from  $(0, 0)$ . (b) Abstract of expand-and-all-pass traversing from  $(-3, 1)$  and  $(3, 1)$ . (c) Example that defects occur at the first row. (d) Conditions of a defect occurring at  $(0, 0)$ .

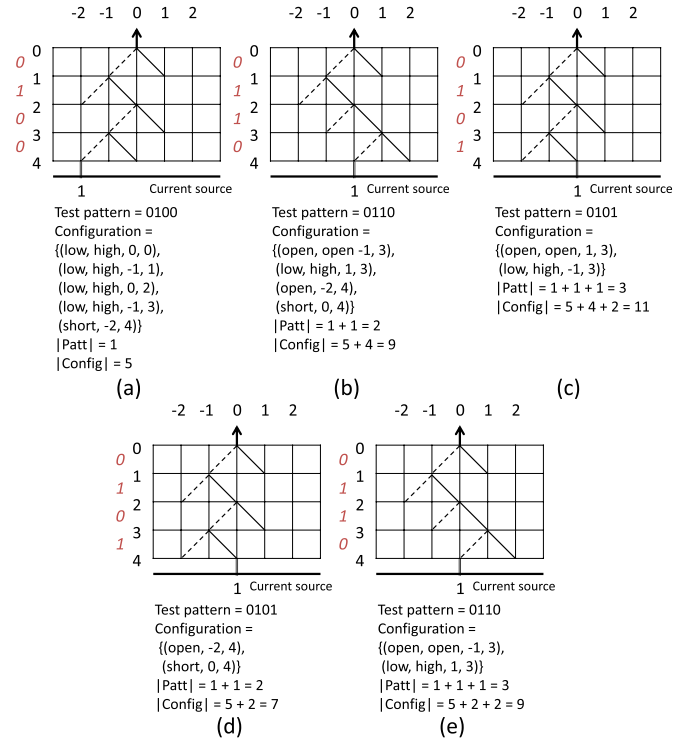


Fig. 13. Examples of the influence of the sequence of path configuration with respect to the required node configurations. (a) Five configurations with the pattern 0100. (b) Additional four configurations with the pattern 0110 after (a). (c) Additional two configurations with the pattern 0101 after (b). (d) Additional two configurations with the pattern 0101 after (a). (e) Additional two configurations with the pattern 0110 after (d).

in Section IV-B). If an open defect is identified at the first row, the diagnosis process expanded outward from the defective node will not further identify any defects. Thus, the edges outside this diagnosis boundary will be marked as unknown edges, as shown in Fig. 12(c).<sup>1</sup> The short defects, however, are harmless to the edges at the first row, except for  $(0, 0)$ , since these nodes are used only for expansion. Thus, we do not identify the short defects occurring at the first row, except for  $(0, 0)$ , in the diagnosis process.

Next, for the node at  $(0, 0)$ , since it is directly connected to the current detector, it is not allowed to suffer from any types

<sup>1</sup>Note that we apply the expansion only for the first row in the proposed static diagnosis. The reason is that the available space for expansion at the first row serves as an indicator of the available mapping width of the given SET array.

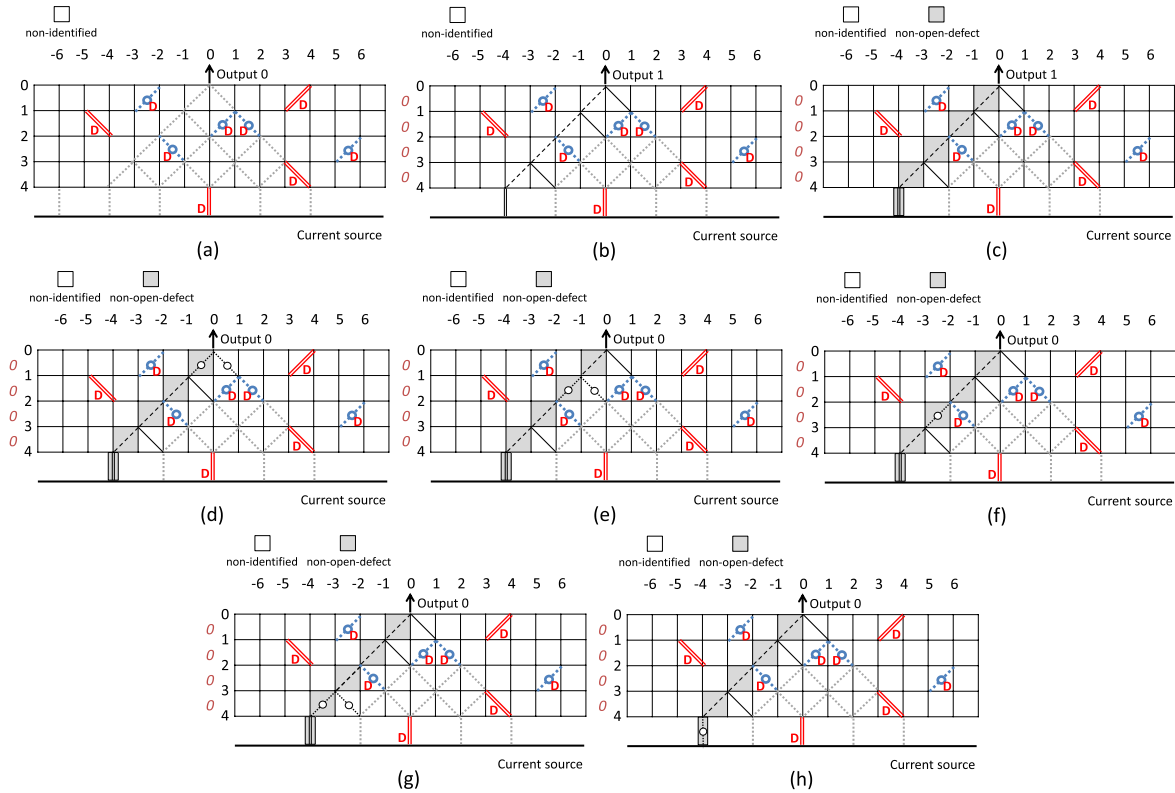


Fig. 14. Examples for demonstrating the process of identifying non-open-defect edges. (a) The defect distribution. (b) A conducting path with the pattern 0000. (c) The non-open-defect edges are marked. (d)(e)(f)(g)(h) (open, open) reconfigurations at (0, 0), (−1, 1), (−2, 2), (−3, 3), and (−4, 4), respectively.

of defects; otherwise, the succeeding mapping process after the diagnosis would fail with a very high probability. Thus, if any defect at the node of (0, 0) is identified, the diagnosis process will be terminated and this SET array will be discarded due to uselessness for mapping. Fig. 12(d) shows all the cases that a defect occurs at the node of (0, 0).

In the expand-and-all-pass method, the order of the path configuration in the diagnosis process influences the required number of node configurations. Here we directly use an example to illustrate this issue. In Fig. 13, the notation (*left, right, x, y*) represents the configuration (*left, right*) of a node device at the location (*x, y*) where (*left, right*) is one of the four operating modes, i.e., (*low, high*), (*high, low*), (*open, open*), and (*short, short*). However, for the configuration of the connection to the current source, we use the notations of (*short, x, y*) or (*open, x, y*).

To configure the path in Fig. 13(a), we have the configurations of (*low, high, 0, 0*), (*low, high, −1, 1*), (*low, high, 0, 2*), and (*low, high, −1, 3*). Note that a *short* configuration (*short, −2, 4*) is also required for connecting to the current source. Thus,  $|\text{config}| = 5$  for this path.

When we configure another path as shown in Fig. 13(b), we have the configuration of (*open, open, −1, 3*), (*low, high, 1, 3*), (*open, open, −2, 4*), and (*short, 0, 4*). Thus, *four* additional configurations are needed. Similarly, to configure the path as shown in Fig. 13(c), *two* additional configurations are required, i.e., (*open, open, 1, 3*) and (*low, high, −1, 3*). Thus, the total number of configurations from Fig. 13(a)–(c) is 11.

In contrast, if we change the order of the path configuration, e.g., exchanging the pattern 0110 with 0101, the required number of configurations would be changed as well. As shown in Fig. 13(d) and (e), if we configure the path in Fig. 13(d) [originally in Fig. 13(c)] first, we have the configurations of (*open, −2, 4*) and (*short, 0, 4*) since the other previous configurations on node devices can be reused. Therefore, this path requires only *two* additional configurations. Finally, to configure the path in Fig. 13(e), (*open, open, −1, 3*) and (*low, high, 1, 3*) configurations are required. The total number of configurations is reduced to 9 from 11. To minimize the required number of configurations in the proposed expand-and-all-pass method, we gradually configure the paths from the left to the right to traverse every path in a subarray.

### B. Example

Fig. 14(a) shows the defect distribution on an SET array where all nodes are initially configured as (*open, open*). Note that the unspecified nodes in the array are all configured as (*open, open*) as well for the ease of demonstration. In addition, the edges in the white boxes of the grid mean that these edges have not been identified yet.

As shown in Fig. 14(b), we start to traverse the subarray rooted at (0, 0) from the leftmost side in the subarray. We can see that there is a path conducted with the pattern 0000 and with the configurations  $\{(low, high, 0, 0), (low, high, −1, 1), (low, high, −2, 2), (low, high, −3, 3), \text{ and } (low, high, −4, 4)\}$ .

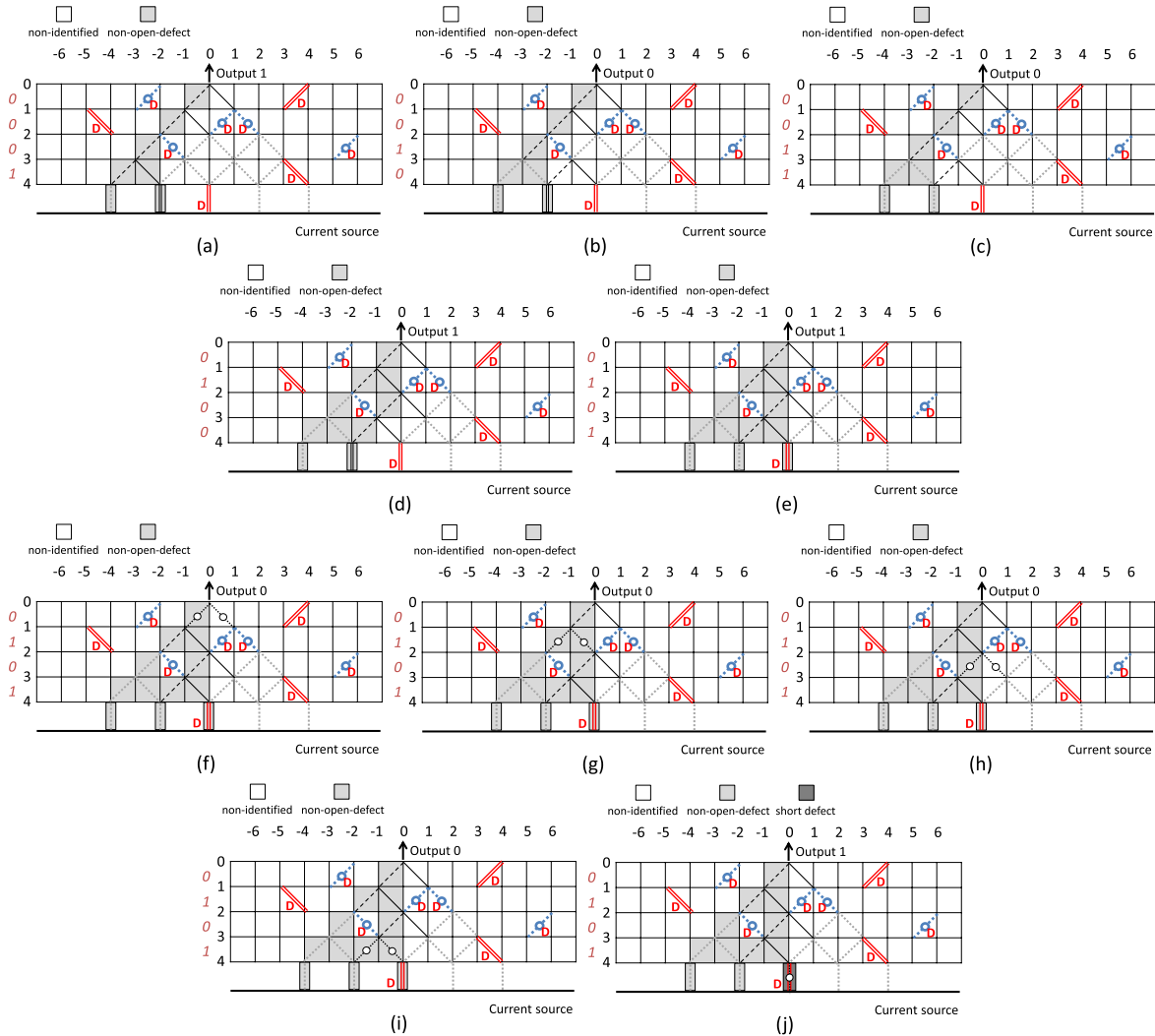


Fig. 15. Examples for demonstrating the process of identifying short-defect edges. (a) A conducting path with the pattern 0001. (b)(c) The paths cannot be conducted with the patterns 0010 and 0011, respectively, due to an open defect. (d) A conducting path with the pattern 0100. (e) A conducting path with the pattern 0101. (f)(g)(h)(i)(j) (open, open) reconfigurations at (0, 0), (−1, 1), (0, 2), (−1, 3), and (0, 4), respectively.

(short, −4, 4)}. Thus, we can know that the edges along the conducting path do not have any open defect. These edges are then marked as *nonopen-defect* edges, as highlighted in the gray boxes of Fig. 14(c). Next, we configure the node at (0, 0) as (open, open), as shown in Fig. 14(d) to check whether the left edge of the node at (0, 0) suffers from a short defect. The SET array outputs 0 since this path is not conducted under the pattern 0000. Therefore, we can conclude that the left edge of the node at (0, 0) does not suffer from a short defect. We continue to configure the lower node, i.e., (−1, 1) as shown in Fig. 14(e), as (open, open) and apply the same pattern 0000; we can conclude that the left edge of the node at (−1, 1) is not a short defect. Similarly, we can conclude that the edges along this path do not suffer from any short defects after these (open, open) reconfigurations, as shown in Fig. 14(f)–(h).

To traverse the next path, we reconfigure the connection of the current source from (−4, 4) to (−2, 4) and apply the pattern 0001, as shown in Fig. 15(a). Since the SET array outputs 1, the connection at (−2, 4) can also be marked

as nonopen-defect edge. After having (open, open) reconfigurations along this path, we can also know that the edges along this path do not suffer from any short defects. Next, as shown in Fig. 15(b) and (c), the next configured paths with the patterns 0010 and 0011 cannot be conducted due to the existence of an open defect at the right edge of the node at (−2, 2). Therefore, no additional nonopen-defect edges are marked.

After checking the conducted path with the pattern 0100 as shown in Fig. 15(d), the path shown in Fig. 15(e) is also conducted with another pattern 0101. Next, we configure (open, open) along the path as shown in Fig. 15(f)–(i) to identify short-defect edges, and no short-defect edges are identified. However, when (open, 0, 4) reconfiguration is set as shown in Fig. 15(j), the path is conducted with the pattern 0101 due to a short defect at (0, 4). As a result, we can assert that the connection at (0, 4) suffers from a short defect.

Fig. 16(a) shows the nonopen-defect and the short-defect edges after checking all the paths in the subarray under the left edge of the node at (0, 0). Since there exists at least one



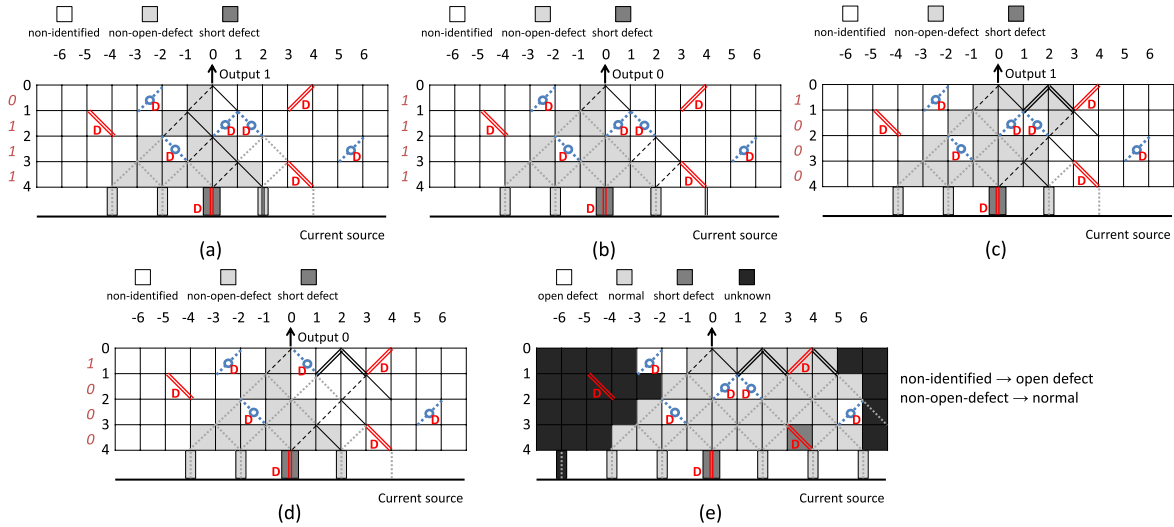


Fig. 16. Examples for demonstrating the method of identifying open defects, unknown edges, and normal edges. (a) A conducting path with the pattern 0111. (b)(c) The steps to identify a double-open-defect at (1, 1). (d) The step to identify a single-open-defect on the right edge of the node at (0, 0). (e) The result of the diagnosis process.

conducting path from the left edge of the node at (0, 0) to the current source in this subarray, we can confirm that the left edge of the node at (0, 0) does not suffer from an open defect. Next, when checking the paths in the subarray under the right edge of the node at (0, 0), we can find that no conducting paths exist along the right edge of the node at (0, 0) as shown in Fig. 16(b). However, we cannot know the exact reason for blocking all the paths. The reason might be either a double-stuck-at-open defect at (1, 1) or a single-stuck-at-open defect on the right edge of the node at (0, 0). To identify which one is the actual reason, we perform the expansion operation from (1, 1) to the right and diagnose the subarray under (3, 1). Since there exists at least one conducting path, as shown in Fig. 16(c), we can confirm that there is no open defect on the right edge of the node at (0, 0). As a result, a double-stuck-at-open defect is identified at (1, 1). On the contrary, if we still cannot configure any conducting path after the expansion from (1, 1) as shown in Fig. 16(d), we can realize that there exists a single-stuck-at-open defect on the right edge of the node at (0, 0).

Finally, Fig. 16(e) shows the results of the diagnosis process for this example. Since the left edge of the node at (-2, 0) suffers from an open defect, the diagnosis process expanded toward the left from (-2, 0) will not further identify any defect. Thus, we mark the edges that cannot be reached as *unknown edges*, highlighted in the black boxes. For the mapping process, the unknown edges will be treated as highly vulnerable edges and cannot be used. Note that the unknown edges occurring at the rightmost side of the SET array result from the ordinary configuration boundary. The final step is to change the nonidentified edges to open-defect ones, and change the nonopen-defect edges to normal ones.

C. Overall Flow

Fig. 17 shows the flow of the diagnosis method. Given the size of an SET array, we initially set all edges as nonidentified

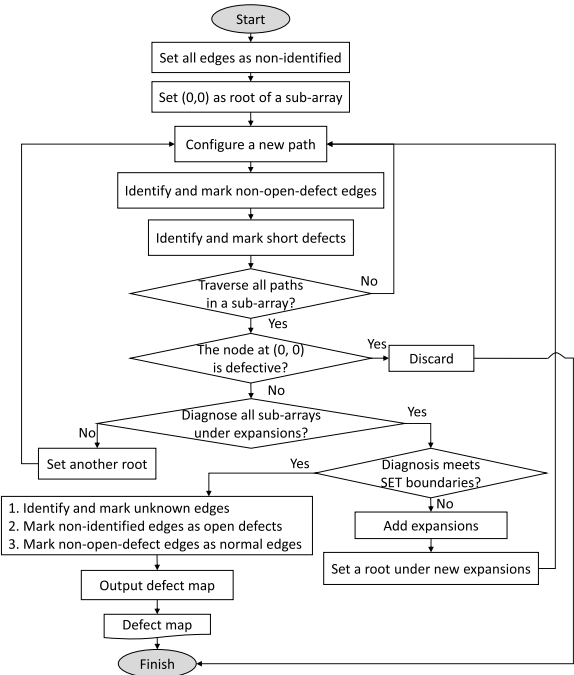


Fig. 17. Flowchart of the proposed diagnosis approach.

and start the expand-and-all-pass from the root at (0, 0). For each path in a subarray, we first identify and mark nonopen-defect edges, and then identify the short-defect edges through (open, open) reconfigurations. After examining all the paths in the subarray, we check if the node at (0, 0) is defective. If the node at (0, 0) is defective, we discard it; otherwise, we expand the subarrays under diagnosis toward the left and right, and continue to diagnose the subarrays under the expansion locations. When the expansions toward both sides meet the boundaries of the SET array, we terminate the process. We then mark the edges outside the diagnosis

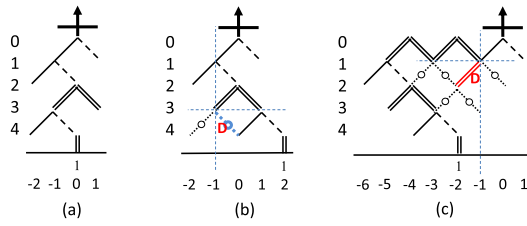


Fig. 18. Examples of baseline detour mapping algorithm. (a) Mapping result of 10-0 without defects. (b) Mapping result when there is a *DOpen* at the right edge of the node at  $(-1, 3)$ . (c) Mapping result when there is a *DShort* at the left edge of the node at  $(-1, 1)$ .

boundary as unknown, mark the nonidentified edges as open-defect ones, and mark the nonopen-defect edges as normal ones.

This algorithm can guarantee 100% defect coverage under the defect distribution assumed in Section III-B. This is important to the succeeding defect-aware mapping algorithm since the incomplete information about defects could result in incorrect results by the defect-aware algorithm. Specifically, the undetected short defects could create invalid paths due to the unexpected connections between paths, and the undetected open defects could block the paths that should be conducted under certain input patterns.

## V. DEFECT-AWARE MAPPING ALGORITHMS

Once the locations of defects on a given SET array have been identified by the diagnosis process, the defect-aware mapping algorithms are able to map a Boolean circuit into the defective SET array. In this section, based on the defect model, we propose two defect-aware mapping algorithms. The first one is the *baseline detour mapping algorithm*, which can successfully map SET arrays by detouring the defective devices. The other one is the *defect-reuse mapping algorithm*, which reuses defective devices for width reduction.

### A. Baseline Detour Mapping

When defects occur in SET arrays, we have to deal with them in the mapping process. Since defects are only *open* or *short*, they can be treated as certain already been configured edges that do not always satisfy the symmetric fabric constraint.

In the baseline detour mapping algorithm, if there is only one *DOpen* edge or *DShort* edge at a node device, we configure the other edge as *open* before mapping. In addition, since the *DShort* edge possibly causes invalid paths, we configure (*open, open*) to node devices that are adjacent to and below the *DShort* edge for isolation. We call these (*open, open*) configurations a *side protection* and a *bottom protection*, respectively.

For example, Fig. 18(a) shows the original mapping result of the product term 10-0. If there occurs a *DOpen* defect at the right edge of the node at  $(-1, 3)$  in the SET array, as shown in Fig. 18(b), we detour the mapping by the short edge and add a (*high, low*) configuration at  $(1, 3)$  for mapping the product term. If there occurs a *DShort* defect at the left edge of the node at  $(-1, 1)$ , as shown in Fig. 18(c), we add a *side*

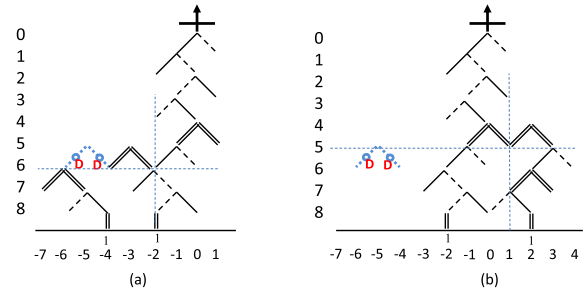


Fig. 19. (a) Expansion meeting defects. (b) Modified expansion.

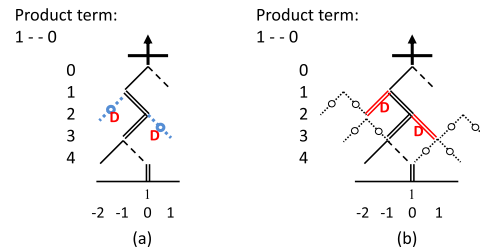


Fig. 20. (a) *DOpen* defects are not on the path to be configured. (b) *DShort* defects are not on the path to be configured.

*protection* and a *bottom protection* at  $(-3, 1)$  and  $(-2, 2)$ , respectively, before mapping the product term. Then, we expand two nodes to the left at the first row to detour the defect, and the original (*high, low*) configuration at  $(-1, 1)$  of Fig. 18(a) is moved to  $(-5, 1)$  in Fig. 18(c). The rest of mapping follows the ordinary mapping algorithm.

In the last examples, we use *short* edges to change the configuration direction or use expansion to detour the defective area for mapping. However, the expansion itself might encounter defective area as well. If the expansion meets defective area, the mapping process looks for another edge at the upper rows for a trial of expansion again. For example, as shown in Fig. 19(a), assume that the product term 1001-100 has been mapped. We would like to map another product term 1001-1-1, and an expansion at  $(-2, 6)$  is required. However, during the expansion to the left for two nodes, we meet a (*DOpen, DOpen*) defect at  $(-5, 5)$ . Therefore, we discard this and expand at another location of  $(1, 5)$  as shown in Fig. 19(b). Finally, the product term 1001-1-1 is successfully mapped.

### B. Defect-Reuse Mapping

Since the defects are *open* or *short*, and they are the same with normal configurations, except singular (one edge) configuration, we can reuse the defects if they are applicable for further width reduction. Therefore, in this section, we propose a defect-reuse mapping approach.

For a single *DOpen* or *DShort* defect occurring at a node device, we leave the other edge intact such that it can be configured as any desired type. Therefore, if the defective edges are not on the conducting path during mapping a product term, we can configure the other edge of the defective node device as usual. For example, Fig. 20(a) and (b) shows valid configurations when *DOpen* or *DShort* defects are not on the path to be configured. We reuse the other edge of a defective node to form the conducting paths.

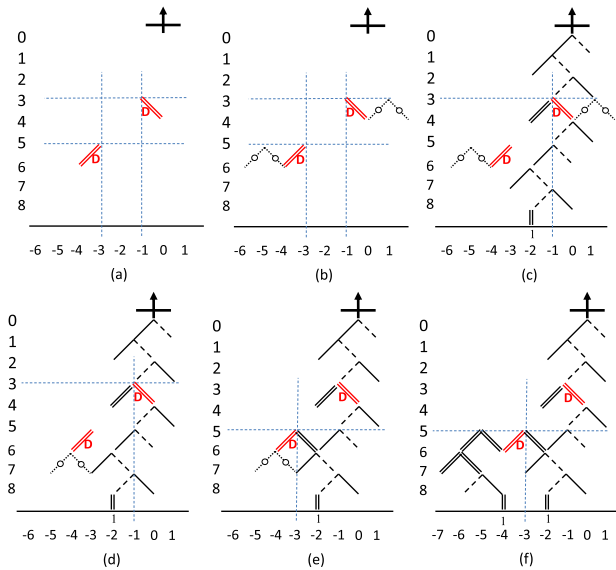


Fig. 21. (a) Two *DShort* edges occur in an SET array. (b) Side protection addition. (c) Mapping result of 100–0100 after reusing the *DShort* at  $(-1, 3)$ . (d) Side protection removal and bottom protection addition. (e) Reusing the *DShort* at  $(-3, 5)$  for expansion. (f) Final mapping result.

In the mentioned baseline detour approach, we isolate all *DShort* edges using the *side protection* and *bottom protection* to avoid creating invalid conducting paths. However, *DShort* edges could be reused when *short* configurations are needed. Therefore, we do not isolate *DShort* edges in the beginning. In fact, the side protection and bottom protection have different purposes. The side protection is used to protect the *conducting path* from short connections to other configurations, which would create invalid paths, when we map a product term downward. However, the side protection would have a side effect that blocks the expansion operation. On the other hand, the bottom protection is used to protect the *expansions* from short connections to other configurations. However, the bottom protection would have a side effect that blocks the mapping path downward. Therefore, to reuse the *DShort* edges without creating invalid conducting paths, we add the side protection of the *DShort* edges only and remove any bottom protection before mapping a product term downward. On the other hand, before expanding the mapping to the left or to the right, we add only the bottom protection of the *DShort* edges and remove any side protection.

For example, assume that there are two *DShorts* in the SET array of Fig. 21(a). Before mapping the product term, the side protections are added at  $(1, 3)$  and  $(-5, 5)$ , as shown in Fig. 21(b). When mapping a product term 100-0100, the *DShort* edge at  $(-1, 3)$  can be reused for the don't care bit of the product term, as shown in Fig. 21(c). For mapping another product term 100-01-1, we need to expand to the left at  $(-2, 6)$ . Before expansion, the side protections at  $(1, 3)$  and  $(-5, 5)$  are removed, and the bottom protection is added at  $(-4, 6)$ , as shown in Fig. 21(d). Note that we do not have to add the bottom protection at  $(0, 4)$  since this location has been configured by the first product term. Fortunately, the *DShort* edge at  $(-3, 5)$  can be reused as well for expansion,

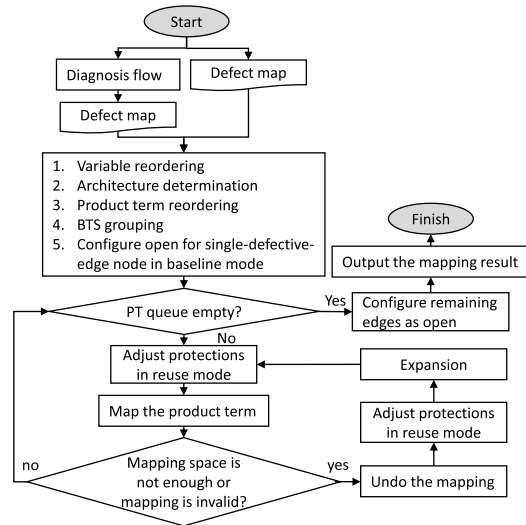


Fig. 22. Flowchart of the proposed defect-aware mapping algorithm.

as shown in Fig. 21(e). To continue mapping the product term from  $(-4, 6)$ , we expand at  $(-5, 5)$ . Finally, the product term 100-01-1 is successfully mapped, as shown in Fig. 21(f), and the unspecified edges are all configured as *(open, open)*.

### C. Overall Flow

Fig. 22 shows the overall flow of the proposed defect-aware mapping algorithm. Given an SET array, we can either perform the diagnosis flow mentioned in Section IV-C to obtain the defect map or directly use a known defect map generated by any source for the defect-aware mapping algorithm. A key point is that the diagnosis process must achieve a high enough defect coverage; otherwise, the correctness of the mapping results by the defect-aware mapping algorithms would be affected. Note that the proposed mapping algorithm can deal with any distribution of defects. However, since  $(0, 0)$  is the location connecting to the current detector, any defect occurring at  $(0, 0)$  would make the mapping process fail. As a result, we assume that the SET arrays having defects at  $(0, 0)$  are discarded before the mapping process, which means that no defect occurs at  $(0, 0)$  in the given SET array. Moreover, if the mapping area exceeds the width of the given SET array or the diagnosis boundary, the mapping process will be terminated.<sup>2</sup>

In the proposed mapping algorithm, the preprocessing procedures including variable reordering, architecture determination, product-term reordering, and BTS grouping are performed first. Before mapping, if the mapping is in the baseline mode, we first configure *open* for the other edge of a node having a single defective edge. Next, for each product term (PT) to be mapped, we first adjust the protections for short defects in the defect-reuse mode. When mapping the product term, if the mapping space is not enough or the mapping process creates an invalid path, we undo the mapping and expand the mapping at a proper row. Note that the protections for short defects will also be adjusted before the expansion in the defect-reuse mode. If the mapping process is

<sup>2</sup>For the ease of discussion, we omit this situation in Fig. 22.

TABLE I  
EXPERIMENTAL RESULTS OF THE PROPOSED DIAGNOSIS METHOD

Size (H×W)	Defect Rate (%)			Coverage (%)	Unknown (%)	Time (s)
	SO	DO	SS			
10×10	2	2	2	100	25.72	<0.01
	3	3	3	100	25.22	<0.01
	4	4	4	100	26.67	<0.01
10×15	2	2	2	100	18.74	0.02
	3	3	3	100	16.78	0.01
	4	4	4	100	20.41	0.01
10×20	2	2	2	100	18.28	0.03
	3	3	3	100	14.04	0.03
	4	4	4	100	18.75	0.03
15×15	2	2	2	100	17.63	1.17
	3	3	3	100	17.95	0.86
	4	4	4	100	19.04	0.81
15×20	2	2	2	100	14.26	1.90
	3	3	3	100	13.59	1.60
	4	4	4	100	18.20	1.36
20×20	2	2	2	100	15.72	58.05
	3	3	3	100	13.99	47.56
	4	4	4	100	17.60	40.63

successful, we proceed to map the next product term. After all the product terms are successfully mapped in the SET array, we configure all the remaining edges that are not configured yet as open and output the final mapping result.

## VI. EXPERIMENTAL RESULTS

The experiments are divided into two parts. The first part shows the validity of the proposed diagnosis flow. In the second part, two sets of experiments were conducted to show the capability of the proposed defect-aware mapping algorithms. We implemented all the algorithms in C language and conducted the experiments on an Intel Xeon E5530 2.40-GHz CentOS 4.6 platform with 64-GB memory.

### A. Diagnosis Algorithm

We conducted diagnosis experiments for different sizes of SET arrays with different defect rates. The defects were randomly injected based on the distribution assumed in Section III-B, and 20 experiments were conducted to obtain the average results. Note that the SET array with a defective node at (0, 0) was discarded and not included in the experiments.

Table I summarizes the experimental results. Column 1 lists the height ( $H$ ) and width ( $W$ ) of the SET arrays. Columns 2–4 list the defect rates of different defect types, single-stuck-at-open (SO), double-stuck-at-open (DO), and stuck-at-short (SS). Column 5 lists the coverage of detected defects. The defect coverage is calculated as (number of detected defects within the diagnosis boundary/number of all defects within the diagnosis boundary)  $\times$  100%. Column 6 lists the ratio of the number of unknown edges with respect to the number of total edges. Column 7 lists the required CPU time. For example, the diagnosis process conducted in an SET array of  $H \times W = 15 \times 15$  with a defect rate of 3% for each defect type reports 100% defect coverage within the diagnosis boundaries. About 18% edges were reported as unknown. The required CPU time under this setting is 0.86 s.

According to Table I, the diagnosis method can achieve a 100% defect coverage under the assumed defect distribution.

TABLE II  
EXPERIMENTAL RESULTS OF THE BASELINE DETOUR MAPPING ALGORITHM AND THE DEFECT-REUSE MAPPING ALGORITHM

Benchmarks	Defect-free		Baseline			Defect-reuse		
	$W_{ori}$	$T(s)$	$W_b$	$TW_b(\%)$	$T(s)$	$W_{dr}$	$TW_{dr}(\%)$	$T(s)$
C17	24	0.05	25.1	4.58	0.09	24.7	2.92	0.09
cmb	35	0.07	41.4	18.29	0.09	40.6	16.00	0.10
cm138a	64	0.06	80.6	25.94	0.08	76.6	19.69	0.08
cm163a	71	0.06	77.0	8.45	0.09	73.6	3.66	0.08
cu	74	0.13	79.0	6.76	0.18	76.4	3.24	0.16
i1	76	0.08	90.4	18.95	0.11	84.0	10.53	0.11
pm1	81	0.07	88.9	9.75	0.09	86.0	6.17	0.09
pcl	89	0.07	101.0	13.48	0.09	94.4	6.07	0.09
x2	89	0.08	94.3	5.96	0.12	93.3	4.83	0.11
cm151a	102	0.06	107.6	5.49	0.09	104.7	2.65	0.09
cm162a	111	0.07	120.9	8.92	0.09	119.4	7.57	0.09
cm85a	112	0.07	120.6	7.68	0.10	118.7	5.98	0.10
cc	132	0.07	140.1	6.14	0.09	135.9	2.95	0.09
unreg	193	0.07	209.4	8.50	0.11	202.8	5.08	0.11
pcler8	218	0.08	239.0	9.63	0.11	231.4	6.15	0.11
c8	220	0.25	242.7	10.32	0.36	239.6	8.91	0.18
cht	313	0.09	334.6	6.90	0.14	326.0	4.15	0.14
count	358	0.11	438.0	22.35	0.17	429.6	20.00	0.18
lal	<b>388</b>	<b>0.10</b>	<b>434.7</b>	<b>12.04</b>	<b>0.14</b>	<b>424.8</b>	<b>9.48</b>	<b>0.16</b>
sct	451	0.08	490.7	8.80	0.12	481.5	6.76	0.14
b9	621	0.14	678.2	9.21	0.22	656.6	5.73	0.30
usb_phy	1117	0.52	1281.0	14.68	0.79	1221.8	9.38	0.86
example2	1200	0.91	1329.5	10.79	1.35	1284.2	7.02	1.49
stepper	2029	0.36	2328.3	14.75	0.55	2256.5	11.21	0.92
sasc	2218	1.32	2434.9	9.78	2.02	2355.8	6.21	2.24
Average	-	<b>0.20</b>	-	<b>11.13</b>	<b>0.30</b>	-	<b>7.69</b>	<b>0.33</b>
Reduction	-	-	-	-	-	-	<b>3.44</b>	-

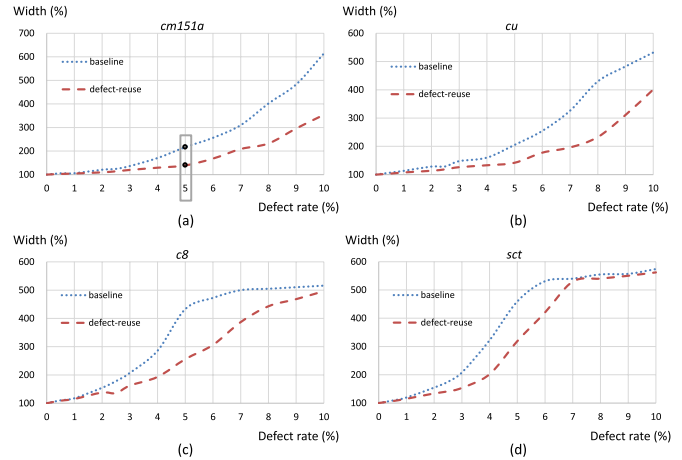


Fig. 23. The comparison of width with different defect rates on benchmarks (a) *cu*, (b) *cm151a*, (c) *c8*, and (d) *sct*, between the baseline detour mapping algorithm and the defect-reuse mapping algorithm.

The 100% defect coverage is important to the defect-aware mapping algorithm since the incomplete information about defects could result in incorrect results by the defect-aware algorithm. The required CPU time for the diagnosis remarkably increases when the height of the SET array increases. This is because the height of the SET array influences the number of paths to be configured. The ratios of unknown edges are varied since the defects were randomly injected. However, when the size of SET array becomes larger, the probability of open defect occurring at the first row near (0, 0) is smaller, which results in smaller ratios of unknown edges.

### B. Defect-Aware Mapping Algorithms

Two sets of experiments were conducted for defect-aware mapping algorithms over a set of IWLS 2005 benchmarks [24]. The first experiment shows the result comparison between baseline detour mapping algorithm and defect-reuse mapping algorithm. The second one shows the width variation with respect to different defect rates between two algorithms. In these experiments, the defects were randomly injected into the SET array with a fixed ratio of failure types. In addition, we conducted the mapping algorithms for ten defect maps of a benchmark to obtain the average results.

In the first experiment, the defect rate was set as 0.5% of the number of sloping edges of the SET array where 0.1% is for stuck-at-short, 0.2% is for single-stuck-at-open, and 0.2% is for double-stuck-at-open.

Table II summarizes the experimental results of the first experiment. Column 1 lists the benchmarks. Columns 2 and 3 list the width of the mapping result and the CPU time for defect-free SET arrays, respectively. Columns 4–6 show the average width of the mapping result, the ratio of the increased width compared with defect-free mapping, and the CPU time in the baseline detour mapping algorithm. Columns 7–9 show the corresponding results in the defect-reuse mapping algorithm.

For example, the original mapping algorithm requires 0.10 s to map the benchmark *lal* on a defect-free SET array with 388 width. The baseline detour mapping algorithm requires 0.14 s to obtain the average result of 434.7 width, or 12.04%  $[(434.7 - 388)/388 \times 100\%]$  increased width compared with the original width. The defect-reuse mapping algorithm requires 0.16 s to map the SET array with 9.48% increased width compared with the original width.

According to Table II, the widths of the mapping results obtained by the defect-reuse mapping algorithm are less than that by the baseline detour mapping with a little CPU time suffering. On average, the ratios of increased width are 11.13% and 7.69% in the baseline detour mapping algorithm and defect-reuse mapping algorithm, respectively. The defect-reuse mapping algorithm reduces 3.44% width on average.

In the final experiment, we map SET arrays with different defect rates. Due to the page limit, we show only the results of four benchmarks, *cu*, *cm151a*, *c8*, and *sct*. Other benchmarks have similar results. Fig. 23 shows the experimental results. The *x*-axis represents the defect rates, and the *y*-axis represents the normalized width of the mapping results. The dotted line and the dashed line represent the widths of the mapping result by the baseline detour mapping algorithm and the defect-reuse mapping algorithm, respectively.

For example, the width of the mapping result by the baseline detour mapping algorithm of benchmark *cm151a* is increased to 216.47% of the original width when the defect rate is increased from 0% to 5%. In contrast, the width of the mapping result by the defect-reuse mapping algorithm is increased only to 138.82% of the original width with the same defect rate, respectively.

According to Fig. 23, we can see that the width was increased when more defects were injected. We can also see that the width difference was getting larger between

the baseline detour mapping algorithm and the defect-reuse mapping algorithm when a higher defect rate was applied. However, for the benchmarks *c8* and *sct*, we can also see that the width difference was getting smaller when the defect rate is higher than a certain value. This is because the distribution of the defects becomes crowded when the defect rate becomes larger, which reduces the probability of reusing the defects by the defect-reuse mapping algorithm. Thus, the width of the mapping result by the defect-reuse algorithm becomes close to that by the baseline detour mapping algorithm when the defect rate is larger than a certain value.

In summary, the defect-reuse algorithm results in less width of the mapping area compared with the baseline detour algorithm among the benchmarks and different defect rates.

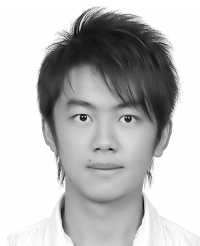
## VII. CONCLUSION

We propose the first approach integrating the diagnosis and mapping of reconfigurable SET arrays. The diagnosis method can achieve 100% coverage of defects under the defect distribution. The mapping algorithms can successfully map SET arrays in the presence of defects. The experimental results show that the defect-reuse mapping algorithm results in less width of the mapping results compared with the baseline detour mapping algorithm.

## REFERENCES

- [1] 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, 2001.
- [2] 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.
- [3] 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.
- [4] 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.
- [5] R. E. Bryant, "Graph-based algorithms for Boolean function manipulation," *IEEE Trans. Comput.*, vol. 35, no. 8, pp. 677–691, Aug. 1986.
- [6] 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.
- [7] 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.
- [8] 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.
- [9] S. Eachempati, V. Saripalli, N. Vijaykrishnan, and S. Datta, "Reconfigurable BDD based quantum circuits," in *Proc. IEEE Int. Symp. Nanosc. Archit.*, Jun. 2008, pp. 61–67.
- [10] L. Liu, V. Saripalli, E. Hwang, V. Narayanan, and S. Datta, "Multi-gate modulation doped  $\text{In}_{0.7}\text{Ga}_{0.3}\text{As}$  quantum well FET for ultra low power digital logic," *Electro Chem. Soc. Trans.*, vol. 35, no. 3, pp. 311–317, 2011.
- [11] 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, no. 3, pp. 415–428, 2010.
- [12] Y.-H. Chen, J.-Y. Chen, and J.-D. Huang, "Area minimization synthesis for reconfigurable single-electron transistor arrays with fabrication constraints," in *Proc. Conf. Design, Autom., Test Eur.*, 2014, pp. 1–4.
- [13] 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. 48th ACM/EDAC/IEEE Design Autom. Conf.*, Jun. 2011, pp. 878–883.

- [14] 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. ID 5.
- [15] C.-E. Chiang *et al.*, "On reconfigurable single-electron transistor arrays synthesis using reordering techniques," in *Proc. Design, Autom., Test Eur.*, Mar. 2013, pp. 1807–1812.
- [16] 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*, Apr. 2015, pp. 1–4.
- [17] C.-W. Liu *et al.*, "Width minimization in the single-electron transistor array synthesis," in *Proc. Design, Autom., Test Eur.*, 2014, pp. 1–4.
- [18] C.-W. Liu *et al.*, "Synthesis for width minimization in the single-electron transistor array," *IEEE Trans. Very Large Scale (VLSI) Syst.*, vol. 23, no. 12, pp. 2862–2875, Dec. 2015.
- [19] Z. Zhao, C.-W. Liu, C.-Y. Wang, and W. Qian, "BDD-based synthesis of reconfigurable single-electron transistor arrays," in *Proc. Int. Conf. Comput.-Aided Design*, 2014, pp. 47–54.
- [20] L. Liu, V. Saripalli, V. Narayanan, and S. Datta, "A defect-aware approach for mapping reconfigurable single-electron transistor arrays," in *Proc. Asia South Pacific Design Autom. Conf.*, Jan. 2015, pp. 118–123.
- [21] L. Liu, V. Saripalli, V. Narayanan, and S. Datta, "Device circuit co-design using classical and non-classical III–V multi-gate quantum-well FETs (MuQFETs)," in *Proc. IEEE Int. Electron Devices Meeting*, Dec. 2011, pp. 4.5.1–4.5.4.
- [22] N. Jha and S. Gupta, *Testing of Digital Systems*. Cambridge, U.K.: Cambridge Univ. Press, 2003.
- [23] L.-T. Wang, C.-W. Wu, and X. Wen, *VLSI Test Principles and Architectures: Design for Testability*. New York, NY, USA: Academic, 2006.
- [24] *IWLS 2005 Benchmarks*. [Online]. Available: <http://iwls.org/iwls2005/benchmarks.html>, accessed May 22, 2014.
- [25] *SET Array Mapping Tool With GUI*. [Online]. Available: <http://nthucad.cs.nthu.edu.tw/~wcyao/>, accessed Jun. 26, 2015.



**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 VLSI designs, and automation for emerging technologies.



**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.



**Chian-Wei Liu** received the B.S. degree from the Department of Computer Science and Information Engineering, National Chung Cheng University, Chiayi, Taiwan, in 2012, and the M.S. degree from the Department of Computer Science, National Tsing Hua University, Hsinchu, Taiwan, in 2014.

Her current research interests include logic synthesis, optimization, and verification for VLSI, system-on-chip designs, and automation in single-electron transistor.



**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.

He has been an Assistant Professor with the Department of Computer Science, National Tsing Hua University, Hsinchu, since 2003, where he is currently a Professor. His current research interests include logic synthesis, optimization, and verification for VLSI/system-on-chip designs and 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.



**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 a member of the Logic Technology Development Group with Intel Corporation, Santa Clara, CA, USA, from 1999 to 2007. 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-k/metal gate CMOS transistors, and the investigation of the transport properties in nonplanar trigate transistors. He was the Joseph Monkowski Associate Professor of Electrical Engineering with Pennsylvania State University, University Park, PA, USA, in 2007, where he is currently a Professor of Electrical Engineering. His group is exploring new materials and novel device architecture for CMOS enhancement and replacement for future energy-efficient computing applications. He holds over 160 U.S. patents.

Dr. Datta is 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 (Penn State), 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 has received several awards, including the Penn State Engineering Society Outstanding Research Award in 2006, the IEEE TRANSACTIONS ON VLSI Best Paper Award at the IEEE Circuits and Systems Society in 2002, the Penn State CSE Faculty Teaching Award in 2002, the Association for Computing Machinery (ACM) Special Interest Group on Design Automation 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 first rank in Computer Science and Engineering in the University of Madras in 1993. He has received several certificates of appreciation for outstanding service from ACM and the IEEE Computer Society. He is the Editor-in-Chief of the IEEE TRANSACTIONS ON COMPUTER AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS.