Received February 22, 2021, accepted March 31, 2021, date of publication April 7, 2021, date of current version April 19, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3071517

On-Chip Error Detection Reusing Built-In Self-Repair for Silicon Debug

HAYOUNG LEE, (Graduate Student Member, IEEE), HYUNGGOY OH, AND SUNGHO KANG, (Senior Member, IEEE)
Department of Electrical and Electronic Engineering, Yonsei University, Seoul 03722, South Korea

Corresponding author: Sungho Kang (shkang@yonsei.ac.kr)

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. 2019R1A2C3011079).

ABSTRACT Post-silicon debug has become important with the increased complexity of circuit designs. However, the increase in debug resource costs owing to improved observability has posed a major challenge. To overcome this challenge, this study proposes on-chip error detection that reuses built-in self-repair (BISR). The proposed method utilizes the components of BISR as storages of golden signatures and comparators for error detection. Also, it detects error-suspect cycles more precisely by using parent and child multiple-input signature registers (MISRs). In addition, it provides selective capture and store methods that selectively capture error-suspect debug data in buffers and store them in the DRAM, respectively. The experimental results of various debug cases demonstrate that the proposed method significantly reduces the buffer size, DRAM usage, and debug time compared to previous methods.

INDEX TERMS Post-silicon debug, error detection, dynamic random-access memory (DRAM) usage, debug time, area overhead, built-in self-repair (BISR), multiple-input signature register (MISR), three-dimensional integrated circuit (3D-IC).

I. INTRODUCTION

With the advance of very large scale integration technology, the density and capacity of integrated circuits has rapidly increased. In addition, to address the increased demands for integration capabilities, three-dimensional integrated circuits (3D-ICs) have been introduced by integrating a system-on-chip (SoC) die and multiple dynamic random-access memory (DRAM) dice with short and dense through-silicon vias (TSVs) [1]–[3]. However, since many components need to be integrated, their reliability is one of the critical issues. To improve the reliability, manufacturing circuit tests using a scan chain or built-in self-test (BIST) are performed to detect faults that may appear during physical implementation. In addition, the practical implementation of built-in self-repair (BISR), which has been widely researched for memory test and repair even before the introduction of the 3D-IC, has been carefully considered for realizing yield improvement and test cost reduction. Its use is possible because the SoC die in the 3D-IC has a large redundant area as it does not have memories owing to the use of multiple DRAM dice.

The associate editor coordinating the review of this manuscript and approving it for publication was Yiming Huo.

Nonetheless, it is challenging to verify or validate these components completely and to reduce the number of errors that remain undetected during pre-silicon verification and manufacturing tests owing to the growing complexity of 3D-ICs. Therefore, post-silicon debug has become an important step in the circuit implementation flow [4], [5].

The main objective of post-silicon debug is detecting errors such as logical, timing, and electrical errors in the first silicon to prevent a cost increase caused by a silicon respin. To observe the maximum feasible internal states of the circuits, the scan-based silicon debug method has been introduced [6]–[9]. It provides a low-cost design for debug (DfD) architecture that reuses scan chains, which are commonly used in manufacturing tests; moreover, it enables to observe a number of internal states that are concatenated by the scan chain. However, the circuit operation needs to be paused to perform a scan dump. Because errors can appear in circuit states during thousands of clock cycles [10], it is challenging to detect these errors in these run-stop debug methods. Therefore, real-time tracing-based debug methods have been introduced. The trace buffer-based silicon debug method is commonly used to observe real-time debug data without pausing the functional operation [5], [11]–[21].

It requires additional on-chip buffers and an embedded logic analyzer to manage trigger points, real-time debug data, etc. The trace buffer-based method is effective for observing real-time debug data for the post-silicon debug; however, its main challenge is the limited observability because the trace buffer size is limited, and it results in DfD hardware overhead. To increase the trace buffer observation, state restoration methods [12], [13] and debug data compression techniques [14]–[18] have been introduced. Nonetheless, trace buffer-based methods still require an exceptionally long debug time. To overcome this limitation, a DRAM-based debug method has been introduced [22]–[24]. In [22], a debug method using external DRAM was introduced and applied to an FPGA prototype; moreover, it will be made available to users of DDR3 and Virtex®-7 or DDR4 and UltraScale®, both on proFPGA hardware. This method enables remarkable improvements in the observability of FPGA prototypes. In [23], a massive signal tracing method using an on-chip DRAM, which can be integrated in the 3D-IC, has been introduced. This method detects erroneous debug intervals using a multiple-input signature register (MISR) and stores the error-suspect debug data dump in the DRAM through the trace and shadow buffer. In [24], a DRAM-based silicon debug method for multiple identical cores has been introduced. This method exploits the fact that the error-free interval data of a core can be applied for the erroneous data of other cores as the golden data. These DRAM-based methods overcome the limitation of trace buffer-based methods. However, they still require substantial debug resources such as buffers and DRAM usage because of the communication between the DRAM and the DfD.

This study proposes on-chip error detection reusing BISR, which is utilized during memory test and repair in manufacturing, for a cycle-accurate deterministic debug environment for a 3D-IC. It is reasonable to assume that BISR can be reused for 3D-IC silicon debug, because the practical implementation of BISR has been highly considered for improving the yield and reducing the test cost of 3D-IC. However, if BIRA hardware exists in 2D-ICs, the proposed method can be applied for the 2D-ICs. In addition, as the proposed silicon debug method reuses BISR, the major problem of BISR being unnecessary after manufacturing can be solved. This is a useful advantage because many studies have failed to solve this problem. The proposed silicon debug method using such conventional BISRs is aimed to improving the quality of error detection by using shorter debug interval detection and reducing the buffer size, DRAM usage, and total debug time. In addition, it is important to note that a cycle-accurate deterministic debug environment is not an impractical assumption [15]–[18], [23]–[28]. Typically, the silicon debug comprises two phases: nondeterministic and deterministic. In the nondeterministic phase, bug occurrences cannot be reproduced because of asynchronous interfaces, interrupts from peripherals, or mixed signal circuitry. In this phase, the main objective is to determine how to regulate the failures. Many studies have aimed to solve this problem [25]–[28]. In the deterministic phase, failures can be regulated by using the abovementioned methods. The main objective of the deterministic phase is to detect the root cause in terms of space (erroneous logic) and time (exact clock cycle when the debug occurs) information as rapidly as possible by using golden data calculated through simulations using the behavioral model of the circuit [15]–[18]. Therefore, the proposed method focuses on a cycle-accurate deterministic debug environment. The main contributions of this paper are as follows:

1) To reduce the debug resource cost, the proposed method reuses BISR. In addition, new architectural features, which can make BISR used for memory repair and silicon debug, are proposed for the DRAM-based DfD.
2) A DRAM-based on-chip error detection method using parent and child MISRs is proposed. As the pre-calculated golden data are stored in the reused BISR, the erroneous debug data can be detected more precisely. In addition, selective debug data capture and store methods are proposed.
3) The DfD operation by reusing BISR is introduced to perform DRAM-based on-chip error detection. It overcomes the challenge of communication between the DfD and the DRAM, which is caused by using the components of BISR as buffers. Also, it reduces the buffer size, DRAM usage, and total debug time.

The remainder of this paper is organized as follows: Section II discusses related works. Section III discusses the motivation for the proposed method and the proposed debug scheme. Section IV presents experimental results for various debug cases. Finally, Section V presents the conclusions of this study.

II. RELATED WORKS

To reduce the silicon debug cost, previous studies have introduced debug schemes that reuse test resources or existing architectural elements [6]–[9], [29]–[31]. These studies reuse test architectures such as the scan chain, test access mechanisms, test bus, IEEE 1149.1, 1500 test wrapper, and/or caches. Since these resources are not utilized after the manufacturing process, debug schemes that reuse test architectures are highly cost-effective in that they can reduce the unavoidable increase in debug resources. Similarly, the proposed method reuses BISR hardware that is not utilized after manufacturing process. In this section, the 3D memory test and repair using BISR and the DRAM-based silicon debug are briefly described.

A. 3D MEMORY TEST AND REPAIR USING BISR

Typically, a 3D-IC consists of an SoC die and multiple DRAM dice. To improve the DRAM yield, the memory test and repair must be performed during the chip implementation process [32]. However, the DRAM test and repair using only external automatic test equipment incurs high test costs because memory tests and repairs in both pre- and post-bond are required, and I/O resources are limited. To overcome this

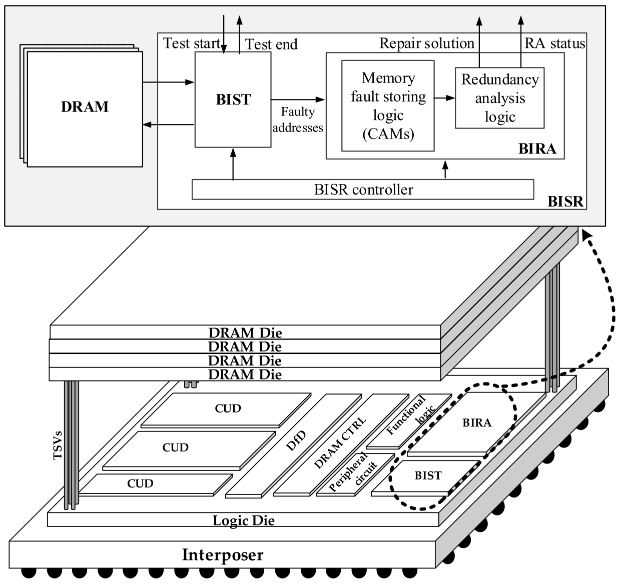


FIGURE 1. Block diagram of memory test and repair using BISR for 3D-IC.

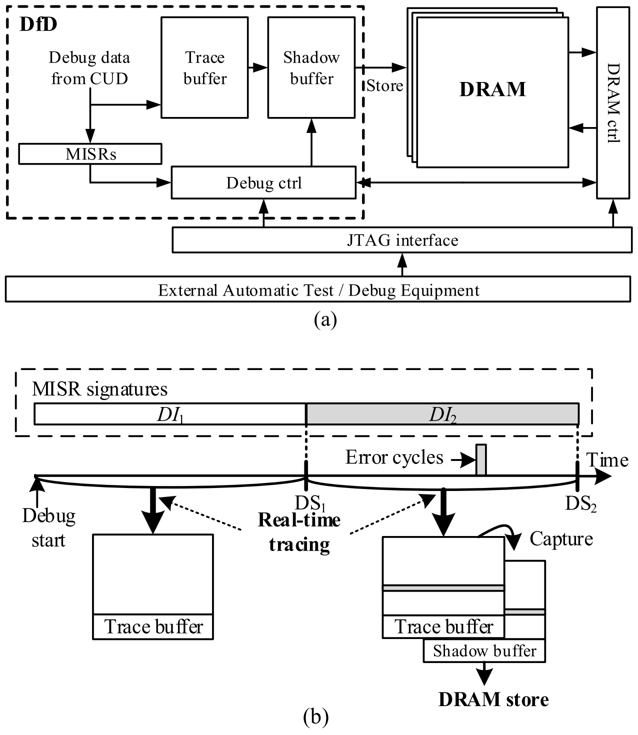


FIGURE 2. The previous DRAM-based silicon debug: (a) block diagram of debug process and (b) example of debug process.

challenge, many studies have investigated BISR [33]–[45]. Fig. 1 shows an overview of the memory test and repair using BISR for the 3D-IC. Typically, BISR consists of the memory BIST and built-in redundancy analysis (BIRA). First, the memory BIST is performed to generate test patterns for memories on each DRAM die and to compare test responses with golden data for identifying the fault sites. Then, the fault information of the memory is sent to BIRA and collected in the memory fault storing logic of BIRA. Simultaneously, the redundancy analyzer of BIRA determines the memory repair solution autonomously. BISR works on the memory test and repair during manufacturing to improve the yield. Therefore, it can be reused after the memory test and repair to reduce the silicon debug resource cost. In the BISR hardware, the proposed method reuses the memory fault storing logic that constitutes a large part of the BISR because it typically consists of content addressable memories (CAMs) since CAM structure has been introduced [46]. The main reason for adopting the CAM structure for the memory repair is that it supports rapid address comparisons between incoming new faults and previously restored faults and stores the information of the faults in a cycle [37]–[41], [43]. Thus, the characteristics of the memory fault storing logic are suitable for silicon debug.

B. DRAM-BASED SILICON DEBUG

The main concept of the DRAM-based method is to transfer the debug data from buffers to a larger on-chip DRAM. Fig. 2(a) shows an overview of the DRAM-based silicon debug process. In [23], only the erroneous interval debug data are captured using the MISR signature. First, golden MISR signatures are generated to detect whether the debug interval is erroneous or not; moreover, they are stored in the DRAM using a trace port such as JTAG. After the debug runs, the debug data are captured in the trace buffer and compressed by MISRs. At the end of the interval, the debug data are analyzed by comparing the MISR signature and the golden signature. Through this debug process, only the debug data of error-suspect intervals are captured in the DRAM. Fig. 2(b) shows an example of the application of the previous DRAM-based method. The main challenge in the previous debug method is that the debug data captured in the buffers are almost entirely error-free, although the corresponding debug session is error-suspect. This is because the error rate of the post-silicon debug is very low. In addition, it is necessary to increase the buffer size and the debug interval detection length to perform the DRAM-based debug method; this is because the DRAM may operate at a lower frequency compared to the debug data sampling frequency. In other words, the debug expense can be reduced significantly by reducing the requirement of debug resources for these error-free debug data and the bottleneck in the communication with the DRAM.

III. PROPOSED DEBUG SCHEME

In this section, the proposed debug scheme that reuses BISR resources is described. First, the motivation for developing the proposed method is discussed by comparing the proposed method to the previous method. Then, the on-chip error detection and the selective debug data capture and store methods using BISR are described. Subsequently, the DfD operation considering the DRAM-based silicon debug method is demonstrated. Finally, the proposed DfD architecture that

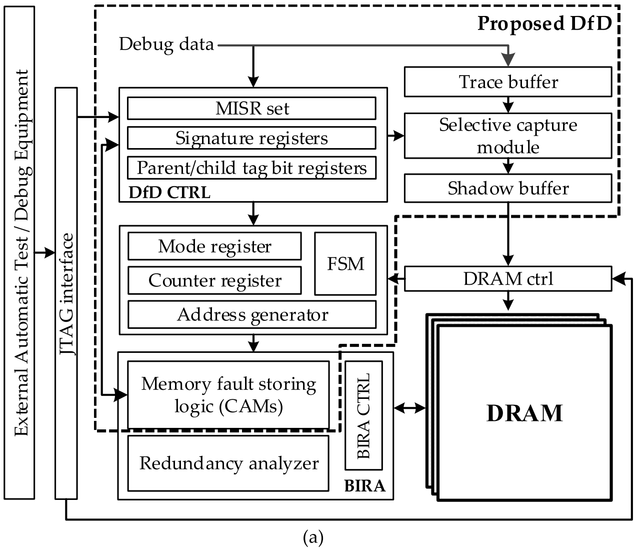
TABLE 1. Notations for debug experiments.

Name	Representation
N	Length of observation window in cycles
M	Trace buffer depth
DS	Debug session (length) in cycles
DI	Debug interval (length) in cycles
L	Number of observed signals
I	Number of debug intervals for DS cycles
SE	Number of segments for a debug interval
P_{max}	Maximum number of golden parent signatures
C_{max}	Maximum number of golden child signatures
GPS	Golden parent signature
GCS	Golden child signature
PS	Parent signature
CS	Child signature
PT	Parent tag bit
CT	Child tag bit
T_{CUD}	Time for silicon debug for the CUD
DU_{CUD}	DRAM usage for the CUD

reuses BISR is described to explain how to reuse BISR. For ready reference, the notations used herein are presented in Table 1; these notations are similar to those used in [23].

A. MOTIVATION

To overcome the challenges faced in the silicon debug processes for 3D-ICs, on-chip error detection method that reuses BISR is proposed in this paper. Notably, the proposed method can be adapted for 3D-ICs, such as the hybrid memory cube (HMC) system [1], TSMC chip on wafer on substrate (CoWosTM) [2], or HBM 2.5D system in package (SiP) [3]. The proposed method imparts several advantages to the DRAM-based silicon debug method. First, BISR is practically suitable for the silicon debug on-chip error detection method because the memory fault storing logic of BISR supports the data comparison required for error detection in a clock given that it is typically constructed with the CAM structure. In addition, the size of the buffers used to communicate with the DRAM can be reduced because the CAM-based memory fault storing logic can be used as buffers pipelining the communication with the DRAM. Consequently, the compression quality can be enhanced for error detection because the debug interval length can be reduced. An overview of the proposed DfD scheme that reuses BISR is shown in Fig. 3(a). As shown, the memory fault storing logic is included in the proposed method. Moreover, the BISR and DfD operations are configured through an external port, for instance, JTAG. Fig. 3(b) shows an example of the proposed method. In Fig. 3(b), *DI* and *DS* represent the debug interval and debug session, respectively. First, the parent MISR signature detects the erroneous interval. Simultaneously, the debug data are periodically compressed to detect the erroneous debug cycles in each session by using the child MISR signatures. These signatures are compared to the golden signatures stored in CAMs. After the debug data are captured in the trace buffer, the erroneous debug data are selectively captured in the shadow buffer and stored in the DRAM. This process



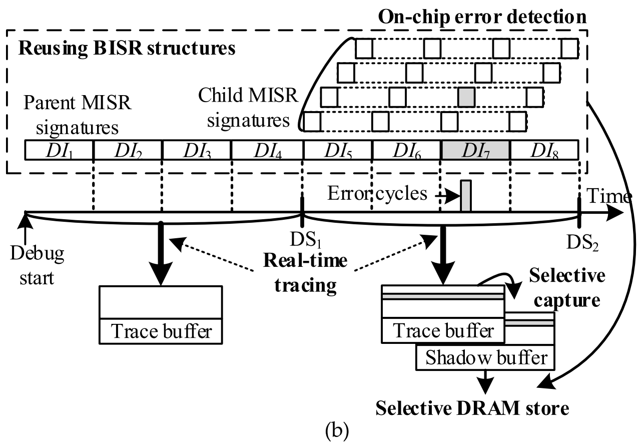


FIGURE 3. The proposed silicon debug. (a) Block diagram of the proposed debug process (b) Example of the proposed debug process.

reduces the use of debug resources, such as buffer size and DRAM usage. The total debug time, which is strongly related to DRAM usage, can be reduced as well.

B. DRAM-BASED ON-CHIP ERROR DETECTION AND SELECTIVE DEBUG DATA CAPTURE AND STORE METHOD

To detect error cycles more precisely, on-chip error detection is performed using hierarchical MISRs during the debug experiment. In the trace-buffer-based silicon debug method, error detection methods that use MISRs have been introduced [15]–[18]. However, the main difference between the existing trace-buffer-based methods and the proposed method is whether the error data detection and capture process are performed in real-time. In the trace-buffer-based methods, multiple debug runs are required to detect and capture the error data because of the limited trace buffer size. However, these processes are conducted simultaneously during only one debug experimental run with the proposed selective debug data capture module and tag bit compression in the proposed method because of the reuse of CAMs

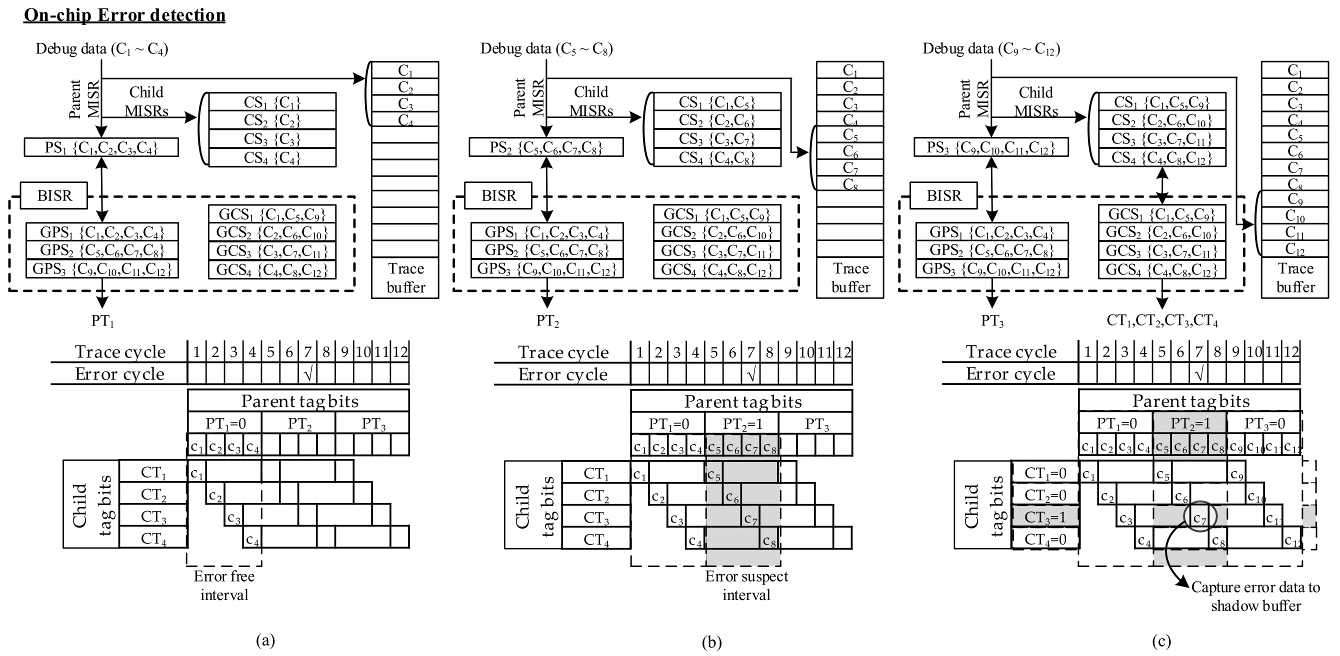


FIGURE 4. Examples of on-chip error detection in the debug case where (a) debug interval in error-free, (b) debug interval is error-suspect, and (c) error cycle is detected using PTs and CTs.

in BISR. In addition, the hardware area overhead is negligible because of BISR reuse, even though the proposed method requires only one debug run, as demonstrated in Section III-D. Consequently, the proposed method provides a 3D-IC silicon debug solution with on-chip error detection by using MISRs and BISR.

To realize on-chip error detection using the CAMs in BISR and the DfD operation, a debug scheduling algorithm is introduced (Algorithm 1). Before the debug experiment, the golden signatures are stored in the CAMs after being transferred from the DRAM. To schedule the period of the golden signatures, this algorithm determines the trace buffer depth (M), total number of debug sessions in cycles (DS), and total number of debug intervals in cycles (DI) by calculating the CAM size (lines 1–5). In the existing DRAM-based method, DI is determined to be larger than the memory access latency because of the speed between the DRAM and the circuit under debug (CUD). Then, DS and M are determined similarly as DI. Unlike the previous method, DI can be reduced in the proposed method because the CAMs can be used as buffers during communication with the DRAM. Thus, M can be reduced as well. The proposed DfD operation considering the communication between the DRAM and DfD is introduced and discussed in Section III-C. In contrast to the method in the previous work, two types of golden signatures are generated in the proposed method. First, the golden debug data are sequentially compressed to detect whether the corresponding debug interval of the debug session is erroneous. These golden signatures are called golden parent signatures (GPS) in this paper. In addition, the golden debug data are periodically compressed to detect the specific error cycles during the debug session. These golden signatures are called golden child signatures (GCS) in this paper. These two golden signatures, namely GPS and GCS, are stored in the DRAM and CAMs during the debug experiment to perform on-chip error detection.

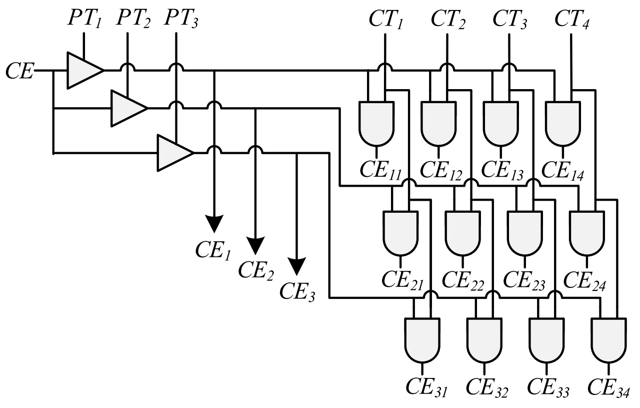
After the start of the debug process, the generated debug data are traced in the trace buffer during the DI cycles. While the debug data are captured in the trace buffer, the debug interval data of each DI cycle are compressed by MISR to detect whether they are erroneous (line 14). In this paper, this MISR for detecting the debug interval is called the parent MISR, and the signature generated by the parent MISR is called the parent signature (PS). In each DI cycle, the PS is compared to the corresponding GPS by using CAMs in a cycle, and the comparison result is obtained as a bit. This bit is called the parent tag bit (PT). If PT is one, the debug data in the trace buffer are captured by the shadow buffer. If not, they are bypassed, and the next debug data are overwritten in the trace buffer (lines 13–19). To detect the error cycles, the debug data are periodically compressed by other MISRs during PS generation (line 21). In this paper, these MISRs for detecting the error cycles are called child MISRs, and the signatures generated by the child MISRs are called child signatures (CSs). These CSs are generated completely during the last DI of DS. Moreover, they are compared to the corresponding GCSs using CAMs by the sequence of a cycle, and the comparison result is obtained as a bit (lines 22–26). This bit is called the child tag bit (CT) in this paper.

Fig. 4 illustrates examples of the on-chip error detection method for each debug case. For clarity, a simple debug case is used, where DS is 12, DI and M are four, I is 3, and

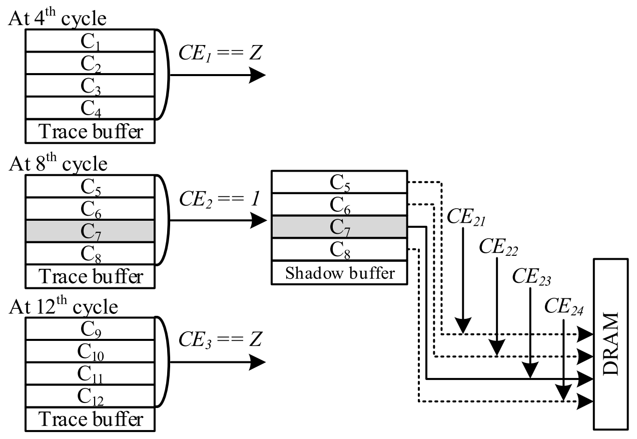
Algorithm 1 Scheduling of the Whole Debug Experiment

```
Input: $I$, $SE$, $L$, $DRAM\ access\ latency$,
Output: $M$, $T_{CUD}$, $DU_{CUD}$
1  Calculate $p$, $c$ using $L$;
2  Update $I$, $SE$ using $p$, $c$;
3  $DS >= DRAM\ access\ latency$;
4  $DI = DS/I$; $M = DI$; Update # of $SB$s;
5  Generate $GPS$s and $GCS$s and transfer to the DRAM;
6  Run debug experiment
7   $i = 0$;
8   while (Debug run) do
9     for each ($DS_i$) do
10      $j = 1; k = 1$;
11      for each ($DI_{ij}$) do
12        Capture the debug data to the $TB$;
13        Generate $PS_{ij}$;
14        if ($PS_{ij} == GPS_{ij}$) then
15          $PT_{ij} = 0$;
16        else
17          $PT_{ij} = 1$;
18          Capture the debug data to the $SB$;
19        while ($k <= SE$) do
20          generate $CS_{ik}$;
21          if ($j == I$) then
22            if ($CS_{ik} == GCS_{ik}$) then
23              $CT_{ik} = 0$;
24            else
25              $CT_{ik} = 1$;
26          $k++$;
27        end
28        $k = 1; j++$;
29      end
30    Store the debug data from the $SB$ to the DRAM;
31    Load the $GS$s from the DRAM to the $GR$s;
32    end
33  end
34 Run post-debug analysis; Calculate $T_{CUD}$ and $DU_{CUD}$;
35 return $T_{CUD}$ and $DU_{CUD}$;
```

SE is 4. In this case, it is assumed that the 7th cycle is the error cycle. After the start of the debug process, the debug data are captured to the trace buffer. Simultaneously, they are compressed using the parent MISR. After the *PS1* is generated, it is compared with the *GPS1* in CAMs, and *PT1* is generated. In this case, *PT1* is zero because this debug interval is error-free. That is, the debug data of the first *DI* (first–fourth cycles) are bypassed. Meanwhile, the debug data are periodically compressed by the child MISRs to generate *CS*s, as shown in Fig. 4(a). After *PT1* is generated, the subsequent debug data are overwritten in the trace buffer and compressed similarly. However, *PT2* is one because the corresponding debug interval is error-suspect. Consequently, the debug data of the second *DI* cycle (fifth–eight cycles) are captured to the shadow buffer, as shown in Fig. 4(b). In the final *DI* (cycles 9–12), the *CS*s and *PS3* are generated, as described in Fig. 4(c). In the ninth cycle, *CS1* is generated and compared with *GCS1* using CAMs. Moreover, *CT1* is generated. In this case, *CT1* is zero because the first, fifth, and ninth cycles are not error cycles. In this manner, the *PT*s and *CT*s are completely generated in the 12th cycle. Furthermore, it can be detected that the 7th cycle is the error cycle. Then, the subsequent debug data are detected on the chip similarly.



(a)



(b)

FIGURE 5. Examples of (a) selective capture module and (b) selective debug data capture process.

To capture the erroneous data selectively, the selective capture module is introduced. The selective capture module for the above example is illustrated in Fig. 5(a). *CE* is the capture enable signal for the shadow buffer. Each of *CEi* and *CEij* are generated using *PTi* and *CTi*. *CEi* and *CEij* are used to select the debug data captured in the shadow buffer and DRAM. In this example, *CE2* is one, whereas *CE1* and *CE3* are high impedance. That is, only the debug data of the second *DI* are captured in the shadow buffer, whereas the debug data of the other cycles are bypassed (lines 15–19). After the debug data are captured in the shadow buffer, the erroneous data are captured in the DRAM by using the DRAM controller and *CEij* signals. In this case, only the debug data of the seventh cycle are stored in the DRAM (line 31) because *CE23* is one, and the others (*CE21*, *CE22*, and *CE24*) are zero, as shown in Fig. 5(b). Consequently, in this example, DRAM usage is reduced by one-twelfth due to the on-chip error detection and selective debug capture method. Moreover, in this simple example, the debug data are captured in cycles. However, in practical debug cases, the clustered debug data detected by parent and child signatures are selectively captured because the observation window (N) is extremely long. The quality of error detection is determined by the size of the reused CAMs in BISR, as demonstrated with various debug cases in Section IV. After the end of the debug experiment, the captured debug data are unloaded to an external workstation, and a post-debug analysis is performed with the result of on-chip error detection. After completion of the debug process, T_{CUD} and DU_{CUD} are calculated and returned at the end of the algorithm (line 35). Then, the subsequent debug plan is performed similarly.

C. DfD OPERATIONS OF DRAM-BASED DEBUG METHODS

In the DRAM-based silicon debug method, it is important to regulate communication with the DRAM during functional operation [22]–[24]. First, a specific DRAM area must be allocated for debug data storage. This is called DRAM usage in this paper. The DRAM usages of the previous and proposed method are discussed in Section IV. Then, the DRAM access operation from the DfD module must be scheduled to prevent interference with the debug experiment. To adapt this operation, adequate sizes of the two buffers (trace buffer and shadow buffer) is required in the DRAM-based debug method. Typically, the size of DS should be larger than the memory read and write latency. DI is equal to DS in the previous method [23] because of the memory read latency for loading the golden signature to the golden register. Thus, M should be equal to DI for capturing the debug data during DI. During DS (DI), the debug data of the corresponding debug interval are captured in the trace buffer. If the corresponding interval is erroneous, they are captured to the shadow buffer. Subsequently, they are stored in the DRAM. This operation is described in Fig. 6(a).

Meanwhile, in the proposed method, the golden signatures in the DRAM can be pipelined to the CAMs in BISR. It is possible because the CAMs usually consist of multiple parts for efficient memory fault storing processes during memory repair processes. An example of the proposed method is shown in Fig. 6(b). In this example, I is 3, and SE is 4. The $GPSs$ are pipelined to the first part, which can be called the parent CAM, at the start of each debug interval. The $GCSs$ are pipelined to the second part, which can be called the child CAM, during the intermediate debug intervals because the CSs are compared to the $GCSs$ in the last debug interval. By using this pipeline operation, DI can be reduced while maintaining the memory access latency. As a result, M can be reduced because DI is reduced. Similarly, as in the previous method, the debug data generated during DI are captured in the trace buffer, and the erroneous interval debug data are selectively captured in the shadow buffer by using PT. In addition, the debug data are periodically compressed to

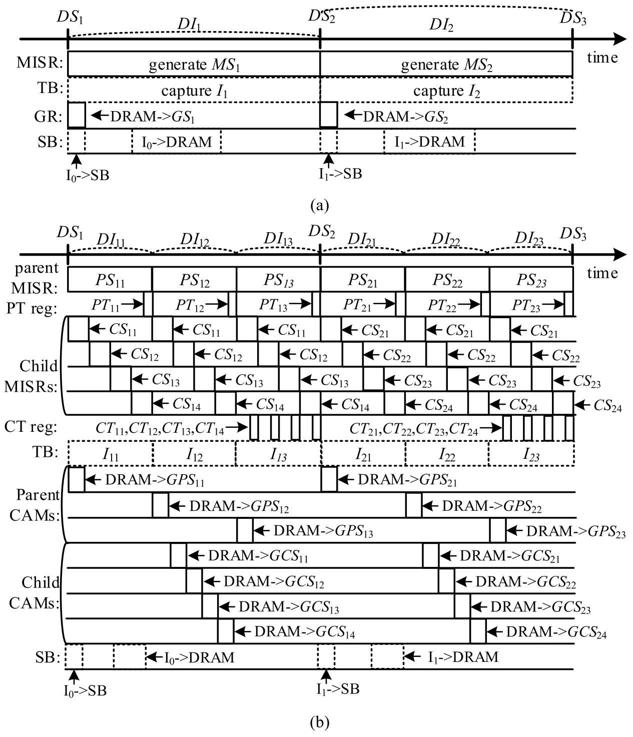


FIGURE 6. Examples of DfD module operation during DRAM-based silicon debug. (a) Previous method and (b) proposed method.

detect the error cycles of each DI through child MISRs. During the final DI of DS, CTs are generated, and the error data can be detected more precisely, as discussed in Section III-B. The error-suspect data in the shadow buffer are stored in the DRAM, and the amount of stored data are reduced compared to that in the previous method because of the selective capture and store method.

D. PROPOSED DfD REUSING BISR

The hardware architecture of the proposed DfD that reuses BISR is illustrated in Fig. 3(a). The BISR and DfD operations are configured through an external port, for instance, JTAG. To handle each process, an additional controller is required, as described in Fig. 7(a). A mode register is implemented to determine the current state. Because the memory repair process is typically carried out before the silicon debug process in chip validation, the mode register is configured such that the current state is the repair phase at first. During the memory repair process, the CAMs and the redundancy analyzer are regulated by the existing BISR controller. After the repair process, the mode register changes the current state in the debug phase. During silicon debug, MISRs are required to generate real-time signatures for on-chip error detection. To perform on-chip error detection during the debug experiment, the golden MISR signatures should be generated by conducting behavioral simulations or emulations on an FPGA board during the configuration step.

In each mode, the address size of the CAMs is calculated using the counter register. Moreover, the address generator is

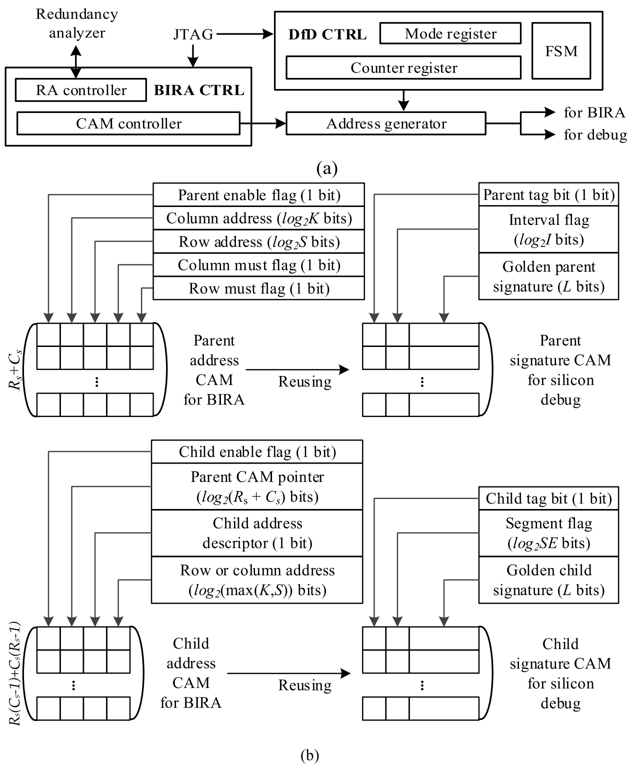

Redundancy analyzer
JTAG
DfD CTRL
Mode register
FSM
Counter register
RA controller
BIRA CTRL
CAM controller
Address generator
for BIRA
for debug
(a)
Parent enable flag (1 bit)
Column address (log_2K bits)
Row address (log_2S bits)
Column must flag (1 bit)
Row must flag (1 bit)
Parent tag bit (1 bit)
Interval flag (log_2I bits)
Golden parent signature (L bits)
R_s+C_s
Parent address CAM for BIRA
Reusing
Parent signature CAM for silicon debug
Child enable flag (1 bit)
Parent CAM pointer ($log_2(R_s+C_s)$ bits)
Child address descriptor (1 bit)
Row or column address (log_2(max(K,S)) bits)
Child tag bit (1 bit)
Segment flag (log_2SE bits)
Golden child signature (L bits)
$R_s(C_s-1)+C_s(R_s-1)$
Child address CAM for BIRA
Reusing
Child signature CAM for silicon debug
(b)

FIGURE 7. Examples of reusing BISR structures. (a) Example of controller for the proposed method. (b) Example of reused CAM structure.

used to adapt the precise size of data for CAMs. These overall operations are regulated by the finite state machine (FSM). In addition, the DfD controller is used for communication between the DRAM controller and the FSM because the golden signatures in the DRAM should be stored in the CAMs during the debug process. The FSM computes the debug cycles and regulates the data transfer between the CAMs and the DRAM. To reuse the CAMs for the golden signatures appropriately, it is important to generate the golden signatures in accordance with the size of the CAMs. Typically, CAMs consist of multiple parts. In Fig. 7(b), examples of CAM structures for BRANCH [37] and the proposed method are illustrated. According to [37], the memory fault storing logic of BRANCH consists of CAMs, and it is used for a $J \times K$ memory block with Rs spare rows and Cs spare columns. The size of the parent CAM is $(3 + \log_2(JK))(Rs + Cs)$ while that of the child CAM is $(2 + \log_2\{(\max(J, K))(Rs + Cs)\})(Rs(Cs - 1) + Cs(Rs - 1))$. In case of the proposed method, the required size of the golden parent (child) signature is $1 + L + \log_2 I$ (SE). In this example, p and c are calculated as

$$P_{max} = \frac{(3+log_2(JK))(R_s + C_s)}{1 + L + log_2 I} \tag{1}$$

$$C_{max} = \frac{(2+log_2\{(\max(J, K))(Rs+Cs)\})(R_s(C_s-1)+C_s(R_s-1))}{1 + L + log_2 SE} \tag{2}$$

where P_{max} and C_{max} denote the maximum numbers of golden signatures during the debug session, and they should be greater than I and SE, respectively. However, as memory faults can occur on all memory cells, all faulty memory addresses should be stored in CAMs for memory repair. Therefore, CAMs are used in spite of large area overhead. Furthermore, the size of CAMs for memory repair is in proportion to Rs and Cs for memory repair, and both Rs and Cs are usually larger than 2 since the density and capacity of memory highly increases. On the other hand, the size of CAMs for the proposed silicon debug can be decided by L, I and SE. L is a small value compared to the total size of CAMs for memory repair and, can be reduced using a spatial compressor (e.g., an XOR tree [15] and [23]). Furthermore, only if I and SE are set to more than or equal to 2, the proposed method can be applied. However, as the size of CAMs for memory repair is very large, P_{max} and C_{max} can also be very large even when I and SE are set to quite large values. For this reason, as the size of CAMs for memory repair is enough to be reused for the proposed method, it is not a problem to apply the proposed method. BRANCH is exemplified because it is the representative BIRA method. The proposed method that reuses BISR can similarly be adapted to any BISR. The golden signatures are required to perform on-chip error detection, while the debug data are captured in the trace buffer. The result of on-chip error detection is obtained as a bit (where one indicates a failure and zero indicates the absence of failure). During the debug experiment, the erroneous data are selectively captured in the shadow buffer and stored in the DRAM by using the tag bits. Then, the subsequent debug data are captured in the trace buffer, and the subsequent golden signatures are transferred from the DRAM to CAMs and stored in CAMs. After the end of the debug process, the validation process ends if the CUD satisfies the product release qualification.

IV. EXPERIMENTAL RESULTS

This section discusses the experimental results in terms of DRAM usage, debug time, and hardware area overhead to describe the benefits of the proposed silicon debug method that reuses CAMs in BISR. In the experiments, DRAM-based silicon debug methods [22] and [23] are used for fair comparisons. Although several silicon debug methods have been proposed after the silicon debug methods [22] and [23], they are not the DRAM-based silicon debug method but the trace buffer-based silicon debug method. Furthermore, there is no study on the DRAM-based silicon debug method after then. It is because the trace buffer-based silicon debug method is widely used for silicon debug. However, if the advantages of the DRAM-based silicon debug method are highly considered, it can also be widely used in the future. It is reasonable since three-dimensional integrated circuits (3D-ICs) have been introduced by integrating a system-on-chip (SoC) die and multiple dynamic random-access memory (DRAM) dice with short and dense through-silicon vias (TSVs). For this reason, we progressed experiments using other DRAM-based

TABLE 2. DRAM usage and debug time comparison with different error rates.

DS	Error rate (%)	DRAM usage (*M* Bytes)						Debug time (*M* cycles)					
		Conv [24]	*Prev* [23]	*Prop*				*Conv* [24]	*Prev* [23]	*Prop*			
				I = 2 *SE* = 2	*I* = 2 *SE* = 4	*I* = 4 *SE* = 4	*I* = 4 *SE* = 8			*I* = 2 *SE* = 2	*I* = 2 *SE* = 4	*I* = 4 *SE* = 4	*I* = 4 *SE* = 8
ARM-based design [47]													
256	0.05		1.09	0.42	0.34	0.35	0.44		86.39	35.60	29.45	29.89	37.59
	0.32		4.382	1.81	1.18	0.86	0.72		353.22	146.59	96.58	70.72	59.69
	0.87		7.08	4.05	2.75	1.92	1.36		569.15	325.71	222.19	156.26	111.02
512	0.05		1.97	0.65	0.40	0.30	0.28		159.71	53.71	34.17	26.00	24.86
	0.32	8.39	5.99	3.09	2.09	1.48	0.97	673.19	481.26	249.56	168.88	119.95	79.17
	0.87		8.11	6.19	3.68	3.61	2.42		650.87	497.53	376.19	290.91	195.34
1,024	0.05		3.45	1.21	0.65	0.43	0.29		278.33	99.25	54.48	36.22	24.90
	0.32		7.14	4.79	3.58	2.80	1.86		572.98	384.79	288.34	225.64	150.66
	0.87		8.36	7.82	6.81	5.99	4.48		671.29	627.76	547.29	481.83	360.55
OpenSPARC T2 [48]													
256	0.10		3.54	1.27	0.93	0.80	0.94		285.21	104.01	76.64	65.97	77.01
	0.32		9.30	3.75	2.43	1.70	1.41		746.42	301.97	196.23	138.09	114.62
	0.87		14.85	9.10	6.28	4.36	3.12		1,190.26	730.28	504.31	350.89	251.77
512	0.10		6.30	2.21	1.34	0.85	0.70		506.40	178.72	109.54	70.09	58.34
	0.32	16.78	13.57	6.90	4.47	3.01	1.90	1344.27	1,087.37	554.16	359.88	242.94	153.82
	0.87		16.57	13.76	10.77	8.34	5.81		1,327.24	1,102.61	863.94	669.68	466.94
1,024	0.10		10.31	4.18	2.50	1.49	0.92		827.20	336.74	202.27	121.13	75.78
	0.32		16.11	11.81	8.32	6.13	3.83		1,290.53	947.21	667.33	492.76	308.58
	0.87		16.77	16.43	14.99	13.60	10.64		1,343.62	1,316.83	1,201.45	1,090.04	853.51

silicon debug methods, [20] and [21]. The experimental results are presented for Ambercore, the ARM-based processor design, [47] and the CPU core in OpenSPARC T2 [48]. To implement DRAM-based debug methods, the CAMs in BISR, buffers, and DRAM are modeled as Verilog modules. The errors in the silicon debug process are randomly injected into the CUD to reproduce misbehavior according to various error rates. A 32-bit data bus is used in the ARM-based design, and a 64-bit data bus is used for the CPU core in OpenSPARC T2. It is assumed that the data bus is utilized for trace signal selection like a lot of related methods.

A. DRAM USAGE AND DEBUG TIME

Table 2 presents the DRAM usages and debug times of the DRAM-based conventional method [22], previous method [23], and proposed method in the debug experiment, where N is $2M$ cycles. As discussed previously, I is the number of *GPS*, and *SE* is the number of *GCS*, and they are determined by calculating the size of the CAMs in BISR. It is determined from the memory and row (column) spare number, *Rs* (*Cs*). Thus, I and *SE* are determined using the assumption of various BISR methods. *DS* indicates debug session, and it is determined by the DRAM read and write latency. DRAM usage is calculated as described in [24]. First, the DRAM usage of the conventional method is calculated as follows:

$$DU_{conv} = L \times N \tag{3}$$

where L is the number of trace signals. Because the error detection process is not executed, all of the debug data are stored in the DRAM in the conventional method.

In the previous method, the error session detection process is performed. First, the expectation of the number of erroneous sessions is calculated as follows:

$$P_{DS} = (1-(1-p))^{DS} \tag{4}$$

$$E[X] = P_{DS}\frac{N}{DS} \tag{5}$$

where p indicates the average probability of occurrence of every cycle error, P_{DS} indicates the error session probability, and $E[X]$ indicates the expected value as the number of erroneous sessions. Given that the previous method only stores the debug data of the erroneous sessions, the DRAM usage of the previous method is calculated as follows:

$$DU_{prev} = E\left[X\right]\left(S + L \times DS\right) + L\frac{N}{DS} \tag{6}$$

where S is the size of the time stamp that identifies the corresponding error interval, and $L \times N/DS$ is the number of golden MISR signatures stored in the DRAM.

In the proposed method, error detection is performed using the parent and child MISRs. The number of error-suspect cycles (*ESC*) in the kth session is calculated as follows:

$$P_{PT_i} = (1-(1-p))^{\frac{DS}{I}} \quad (\text{i} = 1, 2, \ldots, \text{I}) \tag{7}$$

$$P_{CT_j} = (1-(1-p))^{\frac{DS}{SE}} \quad (\text{j} = 1, 2, \ldots, \text{SE}) \tag{8}$$

$$ESC_k = \cup_i^I \, Cand\{PS_i\} \times P_{PT_i} \cap \cup_i^{SE} \, Cand\left\{CS_i\right\} \times P_{CT_i} \tag{9}$$

where $P_{PTi}(P_{CTi})$ is the probability of $PS_i(CS_i)$ being erroneous, which means $PT_i(CT_i)$ is 1, and $Cand\{PS_i\}(\{CS_i\})$ indicates the cycle candidates of each signature. As a result, the DRAM usage of the proposed method can be calculated as follows:

$$DU_{prop} = E[X] \times S + \sum_{1}^{N/DS} ESC_k \times L + (L+1) \times \frac{N(I+SE)}{DS} \quad (10)$$

where $(L + 1) \times N(I + SE)/DS$ is the number of golden signatures (*GPS* and *GCS*) and tag bits (*PT* and *CT*) stored in the DRAM.

To calculate the debug execution time, the concepts of on-chip sampling and communication time are used. The on-chip sampling time is related to the number of clock cycles elapsed between the trigger point and termination of the debug experiment. In the DRAM-based debug method, the sampling time is N cycles because the debug data can be stored in the DRAM during one debug run. The communication time is the time during which the stored debug data are offloaded through the JTAG interface. Typically, the speed of the JTAG interface is relatively lower than that of the CUD. In this experiment, it is assumed that the speed ratio between the CUD and JTAG is 10. In case of the debug time for the DRAM-based method, the debug time is strongly related to the sampling time when the error rate is low. Meanwhile, the communication time dominates the total debug time as the error rate increases. This debug experiment is performed with the different error rates presented in the second column. The error rates are computed as the number of erroneous cycles divided by N. The third and fourth columns present the DRAM usage (*MB*) and debug time (*M* cycles), respectively, of the conventional, previous, and proposed methods.

As illustrated in Table 2, the DRAM usage of the conventional method is high because all the debug data are stored in the DRAM. This is because the error detection in the conventional method is not performed during the debug experiment. Also, the debug time in the conventional method is long by the same reason. In contrast, the DRAM usages of the previous and proposed method are lower than those of the conventional method. Moreover, the debug times of the previous and proposed method are shorter than those of the conventional method because the on-chip error detection is performed during the debug experiment. In [23], only the debug data of the error-suspect intervals are stored in the DRAM. It means that the DRAM usage and debug time decrease with reduction of error rate. Moreover, the DRAM usage and debug time more decrease as *DS* decreases. However, the error detection quality of the previous method decreases significantly as *DS* increases (the error rate is approximately 0.5% or more). In addition, when the error rate is approximately 1%, the DRAM usage and debug time of the previous method are saturated to a similar extent as that of the conventional method because almost all of the intervals are error-suspect. In contrast, although the DRAM usage and debug time of the proposed method increase as *DS* and the error rate increase, the DRAM usage of the proposed method is always less than that of the previous method. Moreover, the debug time decreases correspondingly compared to that of the previous method. For example, compared to the previous method, the DRAM usage and debug time reduction ratio of the proposed method are approximately 46% in [47] and 36% of that in [48] for the least effective debug case, where *DS* is 1,024 and the error rate is approximately 1%. Thus, the proposed method can overcome the limitations of the previous method, where the DRAM usage and debug time increase significantly as the error rate and *DS* increase.

As discussed previously, a specific DRAM area should be allocated for the silicon debug. The observation window (N) of the CUDs can be limited by this DRAM area. This assumption is practical because a large number of CUDs must be debugged during the post-silicon debug process [4]. For example, the DRAM area required in [48] is approximately 4 GB when the conventional DRAM method is used and the number of CUDs is 32. If the allocated DRAM area is smaller than 4 GB, the number of cores should be reduced to satisfy the DRAM specification. As a result, DRAM usage is strongly related to N. Fig. 8 shows the performance comparison of the previous method [23] and the proposed method when length of observation window in cycles (N) is *2M* cycles, debug session in cycles (*DS*) is 256, and the error distribution is uniform. In Fig. 8, the conventional method [24] is regarded as the basis for comparison since it is a cornerstone method for DRAM-based silicon debug. Fig. 8(a) shows the DRAM usage reduction ratio of the previous method and the proposed method compared to the conventional method. As shown, the DRAM usage reduction ratio of the proposed method is always larger than that of the previous method. It is because as the proposed method can divide *DS* into several debug interval in cycles (*DI*s), the size of erroneous interval debug data which are selectively captured can decrease. Also, as the error rate increases, the DRAM usage reduction ratio of the previous method decreases drastically but, that of the proposed method decreases linearly because of the same reason. In addition, the DRAM usage reduction ratio of the proposed method is nearly 100%, when the error rate is small. It means the proposed method is highly effective for silicon debug since the error rate is extremely low in the post-silicon debug which is generally progressed after the pre-silicon verification and manufacturing test. Likewise, Fig. 8(b) shows debug time reduction ratio of [23] and the proposed method compared to [24]. The debug time is highly related with the size of erroneous interval debug data like DRAM usage since the communication time, which is the time to store erroneous interval debug data in DRAM, can be reduced in proportion to the reduction of the size of erroneous interval debug data. For this reason, debug time shows similar tendencies to DRAM usage.

B. HARDWARE AREA OVERHEAD

Table 3 presents the hardware areas of BISR and the DfD controller in the previous method [23] and the proposed method. For fair comparison, all the hardware was synthesized with the Silvaco 45-nm Open Cell Library [50] and the size of each synthesized hardware was represented as the number

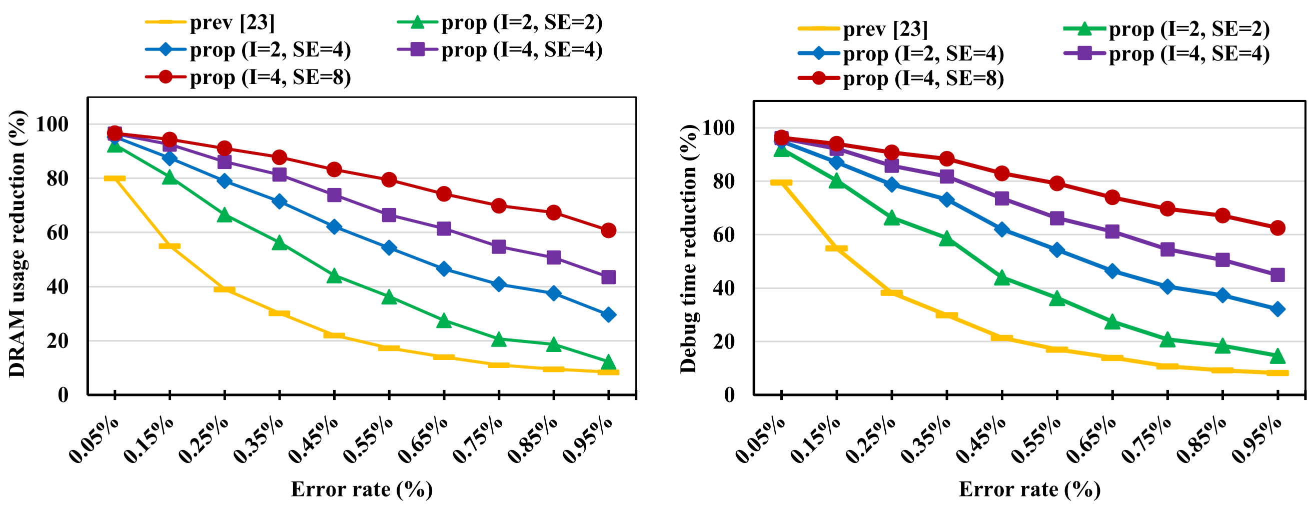


FIGURE 8. **Performance comparison with different error rates. (a) DRAM usage reduction and (b) debug time reduction.**

TABLE 3. **Hardware area overhead comparison.**

Hardware area (2 NAND equivalents)	
spc [48]	BISR (BIRA [37]+memory BIST [49])
204,829	14,603 (7.12%)

DfD modules					
DS	**[23]**	***Prop***			
		$I = 2$ $SE = 2$	$I = 2$ $SE = 4$	$I = 4$ $SE = 4$	$I = 4$ $SE = 8$
256	3,119 (1.52%)	4,660 (2.27%)	5,545 (2.71%)	5,751 (2.80%)	8,571 (4.18%)
512	3,114 (1.52%)	5,784 (2.82%)	6,330 (3.09%)	6,671 (3.25%)	9,291 (4.53%)
1,024	3,106 (1.51%)	7,793 (3.80%)	8,838 (4.3%)	9,593 (4.68%)	11,913 (5.81%)

of two-input-NANDs (NAND2s). In this paper, the BISR consists of memory BIST [49] and a BIRA module [37]. The memory BIST is generated using the Tessent MemoryBIST tool of Siemens. Then, the BIRA module and the BISR controller are designed to adapt the memory BIST by using RTL code. In addition, the DfD modules are also designed in RTL code. The results indicate only the logic circuitry and do not account for the CAMs in BISR or on-chip buffers. Memory BIST and the BIRA module are designed to perform memory tests and repairs for DRAM. The SMarchCHKB algorithm is used to test DRAM in the memory BIST, and the BRANCH algorithm is used to repair DRAM in the BIRA module. The hardware size of the BISR module is determined using the memory size, *Rs*, and *Cs*. In the experiment, it is assumed that the memory size is 1,024 × 1,024, and the value of both Rs and Cs is 2, as in [37]. The CAM size can be increased according to the memory repair case with different *Rs* and *Cs* because it is determined by the memory size, *Rs* and *Cs*. In this experiment, the maximum values of *I* and *SE* are 4 and 8, respectively, as discussed in Section IV-A.

For the silicon debug, the SPARC processor core (spc) module is used. In case of the previous method, two MISRs, a golden signature register, and the interval and time stamp counters are required to perform the debug experiment. Because only the debug intervals are detected using the MISRs, the hardware area overhead of each *DS* is almost identical. The marginal difference between each of the *DSs* can be ascribed to the small size of the debug interval counter. Meanwhile, the proposed method incurs a larger hardware overhead because additional hardware is implemented. In this experiment, the proposed method consists of two parent MISRs and two, four, and eight child MISRs for each *SE*. That is, the hardware area overhead increases as *SE* increases because of the corresponding numbers of child MISRs. Additionally, the selective capture module, address generator module, and counters are used in the proposed method. However, the hardware area of the DfD controller increases as *DS* increases, unlike the previous method. This increment can be primarily ascribed to the fact that the complexity of the selective capture module increases as *DS* increases. That is, the hardware area overhead of the proposed method can be reduced by improving the memory specification. In addition, the size of the address generator module increases as *I* and *SE* increase. Therefore, there exists a tradeoff relationship between the hardware area overhead and the DRAM usage (and total debug time). Nonetheless, the increase of hardware overhead of the proposed method is negligible in Table 3. The hardware overhead in the proposed design increases compared to the previous work [21] in Table 3. However, the overall hardware overhead for silicon debug is decided by the number of used buffers. Indeed, the logic circuitry in the proposed design is less than 20% compared to those in trace buffer-based silicon debug methods [15] and [17]. Buffers occupy more than 80% of the hardware for silicon

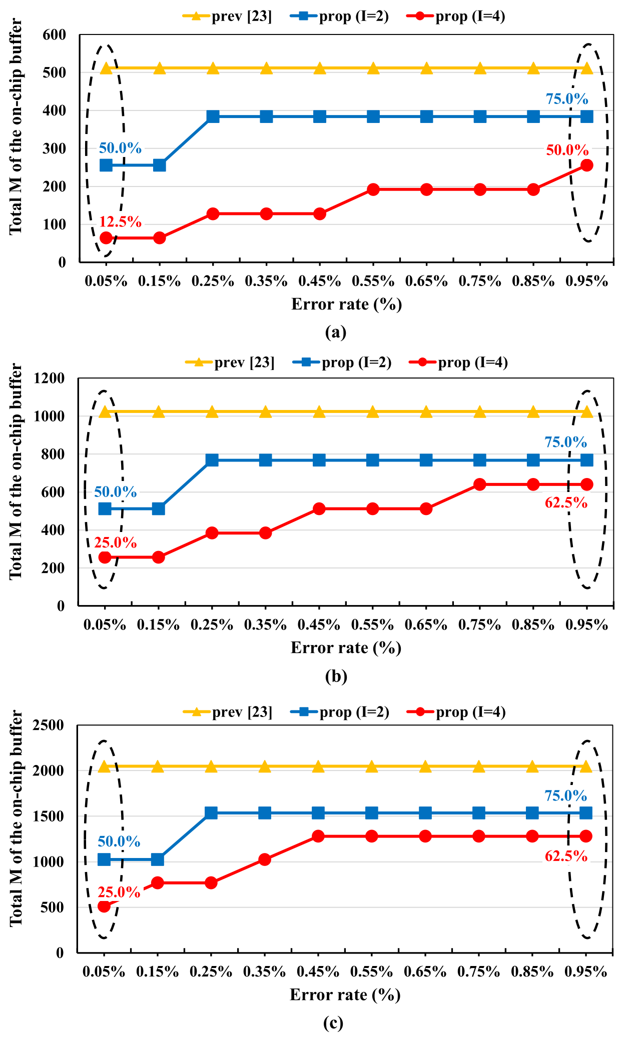


FIGURE 9. Expected results of on-chip shadow buffer size with different error rates. (a) *DS* = 256. (b) *DS* = 512. (c) *DS* = 1,024.

debug when the proposed design is constructed with the worst case in Table 3 (the number of debug intervals for *DS* cycles is 4, the number of segments for a debug interval is 8 and Debug session in cycles is 1,024). For this reason, although the logic circuitry increases up to 5%, it occupies only 1% compared to the total hardware overhead. Thus, the increase of hardware overhead in Table 3 is negligible.

To estimate the size of the on-chip buffers in the previous and proposed methods, *DS* should be determined. As discussed previously, *DS* is determined by the DRAM read and write latency. Moreover, *M* should be equal to *DS* in the previous method to capture the debug data in the trace buffer and shadow buffer during the debug experiment. Thus, the total required buffer depth in the previous method is $2 \times DS$. However, the trace and shadow buffers can be reduced in the proposed method by reusing BISR because BISR can be used to overcome the challenge in the communication between the DfD and DRAM as buffers. First, the trace buffer depth size can be reduced to *DS/I*, which is equal to *DI*. Typically, the size of the shadow buffer should be equal to *DS* because the *CSs* are compared to the *GCSs* in the final *DI* of *DS*. However, in the post-silicon debug phase, where the pre-silicon verification and manufacturing test phase are already complete, errors may occur in certain corner cases such as electrical errors, and the error rate is exceptionally low. Thus, the shadow buffer size can be decreased as the error rate decreases. The shadow buffer depth is estimated by calculating the maximum number of error-suspect *DIs* per *DS* during the entire debug experiment. Fig. 9 shows the on-chip shadow buffer size depending on different error rates and *DS*. In the experiments, *N* is *2M* cycles. Also, *DS* is 256, 512, and 1,024, respectively. As shown in Fig. 9, the shadow buffer size of the proposed method is always smaller than that of the previous method. It is because the smaller error rate is, the smaller the error suspect debug data required to be stored in the shadow buffer is. It results in the reduction of the shadow buffer size. In the same reason, when *I* is 2 in the proposed method, the required shadow buffer size of the error rate smaller than 0.25% is reduced compared to that of other error rates in Fig. 9. However, if *I* becomes large, the shadow buffer size of the proposed method can be more reduced compared to that of the previous method. It is because the error suspect debug data required to be stored in shadow buffer decreases. As shown Fig.9(a), the required shadow buffer size of the proposed method can be reduced by 12.5%, which is the minimum value in the experiments, when *DS* is 256 and the error rate is 0.05%. Although *DS* increases as shown in Fig.9(b) and (c), the required shadow buffer size of the proposed method can be reduced by 25% in the same error rate. In addition, although the error rate increases, the required shadow buffer size can still be reduced by 50% or 62.5%. For this reason, the shadow buffer size of the proposed method can be significantly reduced compared to the previous method.

V. CONCLUSION

In a 3D-IC, the practical implementation of BISR has been extensively considered to improve yield and reduce test costs. Moreover, the DRAM-based debug method has been used to improve the observability of the silicon debug process. In this paper, for the 3D-IC silicon debug process, the DRAM-based on-chip error detection method that reuses BISR is proposed to reduce the DRAM usage, debug time, and on-chip buffer size. Since BISR is unused after the memory repair process, the proposed method can reduce unavoidable increment in debug resources duo to reusing BISR in post-silicon debug. In addition, the proposed debug method can overcome the challenge in the communication between the DRAM and CUD by using BISR as a pipelined buffer. Unlike the previous method, which detects only the error intervals, the proposed method achieves error interval detection and cycle detection by using the CAMs in BISR. The DRAM usage and debug time significantly decrease because of the on-chip error detection. The hardware area overhead of the proposed method is negligible because the sizes of the on-chip buffers decrease when BISR is reused. In addition, the proposed method is compatible for adaptation with other error detection and debugging interconnection techniques with BISR reuse. As a result, the proposed method is suitable for application to practical debug cases of 3D-IC systems.

REFERENCES

[1] J. Jeddeloh and B. Keeth, "Hybrid memory cube new DRAM architecture increases density and performance," in *Proc. Symp. VLSI Technol. (VLSIT)*, Honolulu, HI, USA, 2012, pp. 87–88.

[2] S. K. Goel, S. Adham, M.-J. Wang, J.-J. Chen, T.-C. Huang, A. Mehta, F. Lee, V. Chickermane, B. Keller, T. Valind, S. Mukherjee, N. Sood, J. Cho, H. H. Lee, J. Choi, and S. Kim, "Test and debug strategy for TSMC CoWoS stacking process based heterogeneous 3D IC: A silicon case study," in *Proc. IEEE Int. Test Conf. (ITC)*, Anaheim, CA, USA, Sep. 2013, pp. 1–10.

[3] H. Jun, S. Nam, H. Jin, J.-C. Lee, Y. J. Park, and J. J. Lee, "High-bandwidth memory (HBM) test challenges and solutions," *IEEE Des. Test. Comput.*, vol. 34, no. 1, pp. 16–25, Feb. 2017.

[4] S. Mitra, S. A. Seshia, and N. Nicolici, "Post-silicon validation opportunities, challenges and recent advances," in *Proc. 47th Design Autom. Conf.*, Anaheim, CA, USA, 2010, pp. 12–17.

[5] M. Abramovici, P. Bradley, K. Dwarakanath, P. Levin, G. Memmi, and D. Miller, "A reconfigurable design-for-debug infrastructure for SoCs," in *Proc. 43rd Annu. Conf. Design Autom.*, San Francisco, CA, USA, 2006, pp. 7–12.

[6] X. Gu, W. Wang, K. Li, H. Kim, and S. S. Chung, "Re-using DFT logic for functional and silicon debugging test," in *Proc. Int. Test Conf.*, Baltimore, MD, USA, 2002, pp. 648–656.

[7] B. Vermeulen, T. Waayers, and S. K. Goel, "Core-based scan architecture for silicon debug," in *Proc. Int. Test Conf.*, Baltimore, MD, USA, 2002, pp. 638–647.

[8] R. Datta, A. Sebastine, and J. A. Abraham, "Delay fault testing and silicon debug using scan chains," in *Proc. IEEE Eur. Test Symp. (ETS)*, Corsica, France, 2004, pp. 46–51.

[9] K.-J. Lee, S.-Y. Liang, and A. Su, "A low-cost SOC debug platform based on on-chip test architectures," in *Proc. IEEE Int. SOC Conf. (SOCC)*, Belfast, U.K., Sep. 2009, pp. 161–164.

[10] D. D. Josephson, "The manic depression of microprocessor debug," in *Proc. Int. Test Conf.*, Baltimore, MD, USA, 2002, pp. 657–663.

[11] ARM. (2013). *CoreSight Technical Introduction*. [Online]. Available: http://www.arm.com

[12] H. Fai Ko and N. Nicolici, "Algorithms for state restoration and trace-signal selection for data acquisition in silicon debug," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 28, no. 2, pp. 285–297, Feb. 2009.

[13] X. Liu and Q. Xu, "On multiplexed signal tracing for post-silicon validation," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 32, no. 5, pp. 748–759, May 2013.

[14] E. A. Daoud and N. Nicolici, "Real-time lossless compression for silicon debug," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 28, no. 9, pp. 1387–1400, Sep. 2009.

[15] E. Anis Daoud and N. Nicolici, "On using lossy compression for repeatable experiments during silicon debug," *IEEE Trans. Comput.*, vol. 60, no. 7, pp. 937–950, Jul. 2011.

[16] J.-S. Yang and N. A. Touba, "Improved trace buffer observation via selective data capture using 2-D compaction for post-silicon debug," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 21, no. 2, pp. 320–328, Feb. 2013.

[17] H. Oh, T. Han, I. Choi, and S. Kang, "An on-chip error detection method to reduce the post-silicon debug time," *IEEE Trans. Comput.*, vol. 66, no. 1, pp. 38–44, Jan. 2017.

[18] I. Choi, W. Jung, H. Oh, and S. Kang, "A debug scheme to improve the error identification in post-silicon validation," *PLoS ONE*, vol. 13, no. 9, Sep. 2018, Art. no. e0202216.

[19] H. F. Ko, A. B. Kinsman, and N. Nicolici, "Design-for-Debug architecture for distributed embedded logic analysis," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 19, no. 8, pp. 1380–1393, Aug. 2011.

[20] Y. Cheng, H. Li, Y. Wang, H. Shen, B. Liu, and X. Li, "On trace buffer reuse-based trigger generation in post-silicon debug," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 37, no. 10, pp. 2166–2179, Oct. 2018.

[21] Y. Cheng, H. Li, Y. Wang, and X. Li, "Cluster restoration-based trace signal selection for post-silicon debug," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 38, no. 4, pp. 767–779, Apr. 2019.

[22] D. Amos. (2016). *CertusTM Silicon Debug: Don't Prototype Without*. [Online]. Available: http://s3.mentor.com/fv/volume12-issue1.pdf

[23] S. Deutsch and K. Chakrabarty, "Massive signal tracing using on-chip DRAM for in-system silicon debug," in *Proc. Int. Test Conf.*, Seattle, WA, USA, Oct. 2014, pp. 1–10.

[24] H. Oh, I. Choi, and S. Kang, "DRAM-based error detection method to reduce the post-silicon debug time for multiple identical cores," *IEEE Trans. Comput.*, vol. 66, no. 9, pp. 1504–1517, Sep. 2017.

[25] S. R. Sarangi, B. Greskamp, and J. Torrellas, "CADRE: Cycle-accurate deterministic replay for hardware debugging," in *Proc. Int. Conf. Dependable Syst. Netw. (DSN)*, Philadelphia, PA, USA, 2006, pp. 301–312.

[26] I. Silas, I. Frumkin, E. Hazan, E. Mor, and G. Zobin, "System-level validation of the Intel Pentium M processor," *Intel Technol. J.*, vol. 7, no. 2, pp. 37–43, May 2003.

[27] B. R. Quinton and S. Wilton, "Programmable logic core based post-silicon debug for SoCs," in *Proc. IEEE Silicon Debug Diagnosis Workshop*, May 2007, pp. 1–7.

[28] M. Fujita and H. Yoshida, "Post-silicon patching for verification/debugging with high-level models and programmable logic," in *Proc. 17th Asia South Pacific Design Autom. Conf.*, Sydney, NSW, Australia, Jan. 2012, pp. 232–237.

[29] X. Liu and Q. Xu, "On reusing test access mechanisms for debug data transfer in SoC post-silicon validation," in *Proc. Asian Test Symp.*, Sapporo, Japan, 2008, pp. 303–308.

[30] I. Choi, H. Oh, Y.-W. Lee, and S. Kang, "Test resource reused debug scheme to reduce the post-silicon debug cost," *IEEE Trans. Comput.*, vol. 67, no. 12, pp. 1835–1839, Dec. 2018.

[31] C.-H. Lai, Y.-C. Yang, and I.-J. Huang, "A versatile data cache for trace buffer support," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 61, no. 11, pp. 3145–3154, Nov. 2014.

[32] K. Cho, W. Kang, H. Cho, C. Lee, and S. Kang, "A survey of repair analysis algorithms for memories," *ACM Comput. Surv.*, vol. 49, no. 3, pp. 1–41, Dec. 2016.

[33] S.-K. Lu and C.-H. Hsu, "Fault tolerance techniques for high capacity RAM," *IEEE Trans. Rel.*, vol. 55, no. 2, pp. 293–306, Jun. 2006.

[34] Y. Huang, D. Chang, and J. Li, "A built-in redundancy-analysis scheme for self-repairable RAMs with two-level redundancy," in *Proc. IEEE Int. Symp. Defect Fault Tolerance VLSI Syst.*, Arlington, VA, USA, 2006, pp. 362–370.

[35] S.-K. Lu, Y.-C. Tsai, C.-H. Hsu, K.-H. Wang, and C.-W. Wu, "Efficient built-in redundancy analysis for embedded memories with 2-D redundancy," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 14, no. 1, pp. 34–42, Jan. 2006.

[36] S.-K. Lu, C.-L. Yang, Y.-C. Hsiao, and C.-W. Wu, "Efficient BISR techniques for embedded memories considering cluster faults," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 18, no. 2, pp. 184–193, Feb. 2010.

[37] W. Jeong, J. Lee, T. Han, K. Lee, and S. Kang, "An advanced BIRA for memories with an optimal repair rate and fast analysis speed by using a branch analyzer," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 29, no. 12, pp. 2014–2026, Dec. 2010.

[38] W. Kang, H. Cho, J. Lee, and S. Kang, "A BIRA for memories with an optimal repair rate using spare memories for area reduction," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 22, no. 11, pp. 2336–2349, Nov. 2014.

[39] J. Kim, W. Lee, K. Cho, and S. Kang, "Hardware-efficient built-in redundancy analysis for memory with various spares," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 25, no. 3, pp. 844–856, Mar. 2017.

[40] W. Kang, C. Lee, H. Lim, and S. Kang, "Optimized built-in self-repair for multiple memories," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 24, no. 6, pp. 2174–2183, Jun. 2016.

[41] H. Lee, D. Han, S. Lee, and S. Kang, "Dynamic built-in redundancy analysis for memory repair," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 27, no. 10, pp. 2365–2374, Oct. 2019.

[42] C.-C. Chi, Y.-F. Chou, D.-M. Kwai, Y.-Y. Hsiao, C.-W. Wu, Y.-T. Hsing, L.-M. Denq, and T.-H. Lin, "3D-IC BISR for stacked memories using cross-die spares," in *Proc. VLSI Design, Autom. Test*, Hsinchu, Taiwan, 2012, pp. 1–4.

[43] W. Kang, C. Lee, H. Lim, and S. Kang, "A 3 dimensional built-in self-repair scheme for yield improvement of 3 dimensional memories," *IEEE Trans. Rel.*, vol. 64, no. 2, pp. 586–595, Jun. 2015.

[44] W. Kang, C. Lee, H. Lim, and S. Kang, "A new 3-D fuse architecture to improve yield of 3-D memories," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 35, no. 10, pp. 1763–1767, Oct. 2016.

[45] T. Ni, H. Chang, Y. Yao, X. Li, and Z. Huang, "A novel built-in self-repair scheme for 3D memory," *IEEE Access*, vol. 7, pp. 65052–65059, May 2019.

[46] K. Pagiamtzis and A. Sheikholeslami, "Content-addressable memory (CAM) circuits and architectures: A tutorial and survey," *IEEE J. Solid-State Circuits*, vol. 41, no. 3, pp. 712–727, Mar. 2006.

[47] *Amber ARM-Compatible Core*. Accessed: Dec. 23, 2010. [Online]. Available: http://opencores.org/projects/amber

[48] *OpenSPARC-Based SoC*. Accessed: Oct. 13, 2011. [Online]. Available: http://opencores.org/projects/sparc64soc

[49] *Tessent MemoryBIST*. Accessed: 2019. [Online]. Available: https://resources.sw.siemens.com/en-US/fact-sheet-tessent-memorybist-fact-sheet

[50] *PDK 45nm Open Cell Library*. Accessed: Aug. 1, 2011. [Online]. Available: https://silvaco.co.kr/products/nangate/FreePDK45_Open_Cell_Library/index.html

HAYOUNG LEE (Graduate Student Member, IEEE) received the B.S. degree in electrical and electronic engineering from Yonsei University, Seoul, South Korea, in 2016, where he is currently pursuing the combined Ph.D. degree with the Department of Electrical and Electronics Engineering. His current research interests include built-in self-repair, built-in self-testing, redundancy analysis algorithms, reliability, system-level test and validation, design for testability/debug, and VLSI design.

HYUNGGOY OH received the B.S. and combined Ph.D. degrees in electrical and electronic engineering from Yonsei University, Seoul, South Korea, in 2014 and 2020, respectively. Since 2020, he has been an Engineer with Samsung Electronics Company Ltd., Hwaseong, South Korea. His current research interests include diagnosis, machine learning based failure analysis, design for testability/debug, and system-level test and validation.

SUNGHO KANG (Senior Member, IEEE) received the B.S. degree in control and instrumentation engineering from Seoul National University, Seoul, South Korea, in 1986, and the M.S. and Ph.D. degrees in electrical and computer engineering from The University of Texas at Austin, Austin, TX, USA, in 1988 and 1992, respectively. He was a Research Scientist with the Schlumberger Laboratory for Computer Science, Schlumberger Inc., Austin, and a Senior Staff Engineer with Semiconductor Systems Design Technology, Motorola Inc., Austin. Since 1994, he has been a Professor with the Department of Electrical and Electronic Engineering, Yonsei University, Seoul. His current research interests include VLSI/SoC design and testing, design for testability, design for manufacturability, and fault tolerant computing.