Structured Test Development Approach for Computation-in-Memory Architectures

Moritz Fieback, Mottaqiallah Taouil, Said Hamdioui
Department of Quantum & Computer Engineering
Delft University of Technology
Delft, The Netherlands
m.c.r.fieback@tudelft.nl, m.taouil@tudelft.nl, s.hamdioui@tudelft.nl

***Abstract*—Testing of Computation-in-Memory (CIM) designs based on emerging non-volatile memory technologies, such as resistive RAM (RRAM), is fundamentally different from testing traditional memories. Such designs allow not only for data storage (i.e., memory configuration) but also for the execution of logical and arithmetic operations (i.e., computing configuration). Therefore, not only significant design changes are needed in the memory array and/or in the peripheral circuits, but also new fault models and test approaches are needed. Moreover, RRAM-based CIM makes use of non-linear non-volatile devices making the defect modeling with traditional linear resistor inappropriate for such device defects. Hence, even the way of doing defect modeling has to change. This paper discusses a structured test development approach for RRAM-based CIM and highlights the test challenges and how testing CIM dies is different from the traditional way of testing logic and memory. Methods for defect modeling, fault modeling, and test development will be discussed. The paper demonstrates that unique faults can occur in the CIM die while in the computation configuration and that these faults cannot be detected by just testing the CIM die in the memory configuration. Moreover, it shows that testing the CIM die in the computation configuration reduces the overall test time while improving the outgoing product quality. Finally, the paper presents an outlook on the future of structured CIM test development.**

***Index Terms*—computation-in-memory (CIM), device-aware, defect, fault, test, RRAM, ReRAM**

I. Introduction

Traditional computing systems based on Von Neumann architectures are unable to deliver the computing energy-efficiencies needed for low-power applications that have high data throughput; this is due to, for example, the power-hungry data movement [1]. To overcome this, Computation-in-Memory (CIM) architectures based on emerging memory technologies, such as resistive random access memory (RRAM, or ReRAM) or spin-transfer-torque magnetic RAM (STT-MRAM), are being developed to perform computations directly *within* the memory [2, 3]. This alleviates the need to move large amounts of data, and it thus drastically lowers power consumption. A CIM device can then operate as a standard memory and a computing device. However, in order to perform computations, the standard memory hardware needs to be modified, e.g., the sense amplifier (SA) or address decoder (AD) needs to be redesigned to be able to perform compute operations [2]. These architectural modifications introduce new faults that are not seen in standard memories [4–6]. Furthermore, it has been shown that emerging memory devices suffer from new defects and failure mechanisms that cannot be detected by standard memory tests [7–9]. Hence, in order to realize high-quality CIM, appropriate test solutions are required. These solutions need to take into account both the unique properties of the architecture and those of the devices forming the core of CIM.

There is only a limited work published on testing of CIM. Several researchers have studied the impact of resistive defects (e.g., an open connection, or a short-circuit to GND) on the performance of a CIM architecture and subsequently developed tests to detect such modeled defects [4, 10–13]. The work has shown that there are unique computation faults that are not seen in regular memories. The proposed test solutions consist of modified March algorithms that make use of both configurations. In [14], a test for neuromorphic CIM is developed based on reference trimming. Finally, in [5, 6] fault models and tests for RRAM-based Scouting logic CIM are developed that take into account the unique nature of defective memory devices. Clearly research on CIM testing is still in an early stage. For instance, although there are some approaches to structurally testing of CIM circuits, there is not yet one unified structured test development approach available.

This paper presents a structured device-aware test development approach for CIM architectures. This approach results in tests that test both the memory and computation configuration of the CIM device in a structured manner, while also taking into account the unique defects that can occur in emerging memory devices. In short, this paper contributes the following:

- Structured device-aware test development approach for CIM architectures.
- Validation of the approach on a RRAM-based CIM architecture.
- Outlook on the future of CIM testing.

The remainder of the paper is structured as follows. Section II presents background information on CIM and RRAM. Section III explains the device-aware test development approach. Section IV presents the device-aware defect modeling. Section V presents the fault modeling and analysis. Section VI presents the test development for the CIM architecture. Finally, Section VII presents an outlook on the future of CIM testing and concludes the paper.

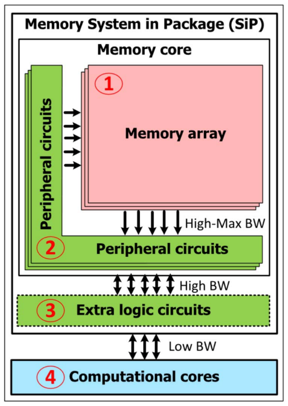
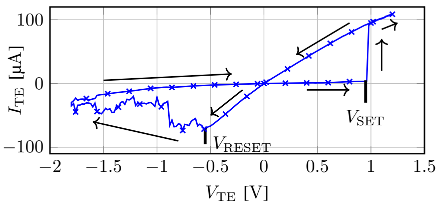
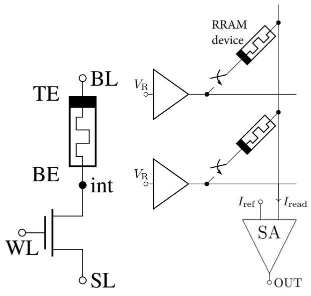


Fig. 1: Computer architectures [15]



(a) RRAM switching



(b) 1T1R cell (c) Scouting logic [2]

Fig. 2: RRAM switching, cell, and usage in Scouting logic

II. Background

This section classifies different computer architectures while highlighting CIM; thereafter it details RRAM-based CIM.

A. Computer Architecture Classification

Computer architectures can be classified based on where the computation takes place with respect to where the data is stored; i.e., either within the memory core/IP or outside it. In [15], the authors presented Fig. 1 that shows four possible locations where computation can take place, resulting into two classes, each with two sub-classes: Computation-in-Memory (CIM) with two sub-classes ① and ②, and computation-outside-memory (COM) with the two sub-classes ③ and ④. The higher the number, the farther away from the memory the computation takes place, which leads to lower bandwidth (BW), longer delays, and more energy consumption.

In ① the computation takes place directly in the memory array and it is typically referred to as CIM-A, while in in ② the computation takes place in the peripheral circuits of the memory device (e.g., by using dedicated sense amplifiers) and it is referred to as CIM-P [15, 16]. In general, CIM-A requires major changes to the cell array design and minor changes to the peripheral circuits, while CIM-P typically requires the modification of the peripheral circuits only! CIM-A/P can be further extended by taking the nature of the operands into consideration; e.g., for two operands CIM-P, if both operands are stored as resistance (r) in the memory array, then it is named CIM-Pr (as is the case for scouting logic [2], which is the focus of this paper); and if one operand is stored as resistance and one is fed to CIM die as voltage then it is named CIM-Ph (as is the case for vector dot product [17]). It is worth noting that a CIM chip can operate as a regular memory in its *memory configuration (MC)*, or as a computing device in its *computation configuration (CC)*. To be able to do this, a regular memory IP needs to be modified. On the other hand, in ③ the computation takes place in additional hardware placed near the memory (e.g., today's commercialized High Bandwidth Memories HBM); and in ④ the computation takes place outside the memory in a dedicated computational core (e.g., today's commercialized CPUs).

B. RRAM-based CIM

A RRAM device is a stack of an oxide between two electrodes (the top electrode (TE) and bottom electrode (BE)). By applying a positive voltage on the TE so that $V_{TE} > V_{SET}$, the bonds between the oxygen and metal ions break, and the oxygen ions are attracted to the TE. This is called a SET operation. This chain of oxygen vacancies is called a conductive filament (CF) that can carry a current. Now, by applying a negative voltage to the TE so that $V_{TE} < V_{RESET}$, the oxygen ions move back into the oxide and disrupt the CF. This is called a RESET operation. Fig. 2a shows the resulting current-voltage graph of this switching process. The resistance of the RRAM device in SET is lower than in RESET and is, respectively, logically interpreted as '1' and '0', as shown in Fig. 3. The state of the RRAM device can be found by applying a low read voltage to the device and comparing the resulting current to a reference in the SA. The resistance of a RRAM device is analog, as such, it is possible that the device has a resistance between '1' and '0'. This state is called the undefined state and is indicated in Fig. 3 with 'U'. Similarly, it is possible that the resistance is below '1' ('0'), this is called the extreme high (low) state, indicated by 'H' ('L') in Fig. 3. Fig. 2b shows a typical one transistor (1T) one RRAM device (1R) memory cell. The cell can be accessed by opening the transistor through the word line (WL) and applying appropriate voltages to the bit line (BL) and select line (SL).

In Scouting logic, AND and OR logic operations can be performed on two cells in the same column. It works by reading two cells at once and comparing the current to a reference that depends on the operation that is performed in the SA, e.g., reading '1' and '0' while setting the reference to OR should output an '1'. Fig. 3 shows the relative resistance of

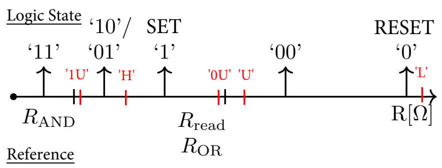


Fig. 3: Scouting equivalent resistance and references

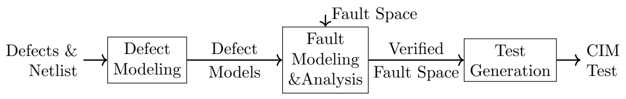


Fig. 4: Device-Aware Test development flow [18]

these references ($R_{\mathrm{read/OR}}$ and R_{AND}) for the two operations as well as the equivalent parallel resistance of the two cells for every logic state (e.g., '11', or '00'). The figure also shows in red the resistance of the faulty states 'L', 'H', and 'U', as well as '1U' ('1' parallel with 'U') and '0U' ('0' parallel with 'U'). Note that different hardware is used in both configurations; in the MC only the R_{read} reference is used, while the R_{AND} reference is only used in the CC. Furthermore, in order to access two cells at once, a second address decoder needs to be included as well.

III. Testing Computation-in-Memory

We apply the Device-Aware Test (DAT) approach [18, 19] to develop tests for CIM architectures making use of scouting logic. DAT does not assume that a defect in a device (or a cell) can be modeled electrically as a linear resistor (as the state-of-the-art approach suggests), but it rather incorporates the impact of the physical defect in the technology parameters of the device and thereafter in its electrical parameters. Once the defective electrical model is defined, a systematic fault analysis is performed to derive appropriate fault models and subsequently test solutions. In our previous work, we have applied the DAT approach to test industrial RRAMs and STT-MRAMs and have demonstrated that DAT sensitizes realistic faults as well as new unique defects and faults that can never be caught with the traditional approaches [8, 9, 20, 21]. In addition, the results have shown how powerful is DAT for fast diagnosis and yield learning.

The DAT approach consists of three steps: 1) defect modeling, 2) fault modeling and analysis, and 3) test generation, and is illustrated in Fig. 4 [18].

1) Defect Modeling: This first step is the core of the DAT approach. Here, the actual physics of a defective device is modeled and incorporated into a compact and calibrated model that can be simulated in a netlist. This is done in three steps: 1) physical defect modeling, 2) electrical defect modeling, and 3) calibration. First, the defect's effect on the physical technology parameters (e.g., length, width, doping density, etc.) of the device is modeled. This results in the defective technology parameters for a device. Second, the defective technology parameters are incorporated into an electrical model that can be used in a SPICE simulator. This step allows studying the effects of the defect on the electrical parameters of a device

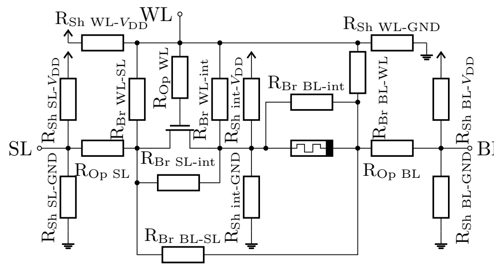

$\mathrm{R_{Sh\ WL\text{-}}}V_{\mathrm{DD}}$
WL
$\mathrm{R_{Sh\ WL\text{-}GND}}$
$\mathrm{R_{Sh\ SL\text{-}}}V_{\mathrm{DD}}$
$\mathrm{R_{Br\ WL\text{-}SL}}$
$\mathrm{R_{Op\ WL}}$
$\mathrm{R_{Br\ WL\text{-}int}}$
$\mathrm{R_{Sh\ int\text{-}}}V_{\mathrm{DD}}$
$\mathrm{R_{Br\ BL\text{-}int}}$
$\mathrm{R_{Br\ BL\text{-}WL}}$
$\mathrm{R_{Sh\ BL\text{-}}}V_{\mathrm{DD}}$
SL
BL
$\mathrm{R_{Op\ SL}}$
$\mathrm{R_{Br\ SL\text{-}int}}$
$\mathrm{R_{Op\ BL}}$
$\mathrm{R_{Sh\ SL\text{-}GND}}$
$\mathrm{R_{Sh\ int\text{-}GND}}$
$\mathrm{R_{Sh\ BL\text{-}GND}}$
$\mathrm{R_{Br\ BL\text{-}SL}}$

Fig. 5: All transistor and interconnection defects

(e.g., the resistance, threshold voltages, etc.). Third, the model is calibrated to match the measurements of the defective device to ensure that a high-quality defect model is obtained.

2) Fault Modeling and Analysis: In this second step, the complete fault space is verified, resulting in the complete set of faults that can occur in the circuit and which need to be detected by a test. To do this, first, the fault space must be defined. Then, the defect models from the previous step are incorporated into a netlist and simulated. The resulting faults are listed and form the verified fault space for which a test needs to be developed.

3) Test Generation: In the third step, a test is generated that can detect all the verified faults. These tests can be march tests that are adapted for the specific faults. However, it is also possible that specialized design-for-test (DFT) structures are needed to guarantee the detection of all faults.

The core idea for applying DAT to CIM testing is to identify which components in a normal memory are modified to allow for CIM capabilities. Because the hardware used in both configurations is not the same, the approach must be applied to both configurations [6]. In the MC, the components that are used for regular memory operations are tested; for instance, the memory array, the SAs in the read configuration, and row/column Address Decoders (ADs). In the CC, the components that are used for compute operations are tested; for instance, the memory array, the SAs in the compute configuration (AND, OR), and the row address decoders as they perform simultaneous access of two operands.

IV. Device-Aware Defect Modeling

We perform defect modeling for both CIM configurations..

A. Memory Configuration

RRAM-based CIM architectures consist of transistors, interconnections, and RRAM devices. Defects in transistors and interconnections are typically modeled as linear resistors because they affect the resistance of a connection [22]. All possible defects in a single 1T1R cell are shown in Fig. 5. There exist three kinds of defects here: shorts (Sh), bridges (Br), and opens (Op). A short defect is a connection between a node and a supply node, e.g., a connection between the BL and GND. A bridge is a connection between two nodes in the circuit that are not supply nodes. An open is an increased resistance within a line, e.g., an increase of the line resistance.

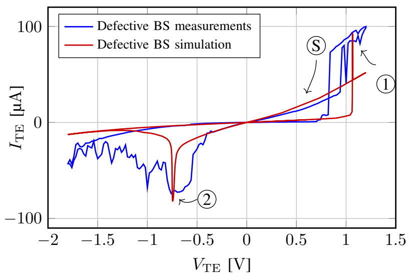


Fig. 6: IUSF in a RRAM device. Measurements and model [9]

In a similar way, these resistive defects manifest themselves in ADs and SAs.

On the other hand, it has been demonstrated using industrial devices that linear resistors fail to describe defects *in* the RRAM device and their non-linear behavior [9, 20]. Therefore, DAT defect models are needed that adequately describe the defective behavior of the RRAM device.

In the rest of this section, we will briefly illustrate DAT defect modeling for two RRAM defects/faults: forming defects [20] and intermittent undefined state fault (IUSF). [9]. Forming defects occur when the current through a RRAM device that forms the initial CF is too high or too low, resulting in devices switching into the 'H', 'L', and 'U' states; see also Fig. 3. The IUSF is caused by a reduced oxygen ion binding capability in the RRAM device, which causes the device to intermittently change its switching mechanism from bipolar to complementary switching. This leads to the device going into 'U' rather than '1' when performing a SET operation. Note that this fault does not happen in every cycle; its behavior is intermittent. Fig. 6 shows the current-voltage graph of the IUSF as it was measured, as well as the fitted and optimized defect model; the figure shows that the device switches from '0' to '1' at ①, but then immediately switches into 'U' at Ⓢ. A subsequent RESET operation first switches back to '1' before it resets to '0' at ②.

B. Computation Configuration

The hardware components forming the CC of CIM are similar to those forming the MC of CIM; these consist of transistors, interconnections, and RRAM devices. Hence, the potential defects in CC are similar to those in MC; the resulting defects models in MC can be used for CC.

V. Device-Aware Fault Modeling

Next, we perform fault modeling and analysis.

A. Memory Configuration

In the MC, we first define the complete memory fault space. Then, we validate it.

1) Fault Space Definition: In the MC, faults can occur in components contributing to the MC of the CIM chip. We will restrict ourselves to the three main components: the memory array, the ADs, and the SAs. We define the fault space for each of these components.

a) Memory Array: Memory array faults are typically described using a fault primitive (FP) concept $\langle \mathrm{S/F/R} \rangle$ [23]:

- S denotes the sensitizing sequence that is applied to the memory cell. These can be write (w) and read (r) operations. For example, writing a '0' to a cell that initially stores a '1' is denoted as 1w0, and reading a cell that contains a '1' is denoted as 1r1. It is possible to have multiple operations in S, e.g., $\mathrm{S} = 0\mathrm{w}1\mathrm{r}1$.
- F denotes the final state of the cell after the sensitizing sequence is applied. For the RRAM device in this paper, it holds that $F \in \{0, 1, \mathrm{U}, \mathrm{H}, \mathrm{L}\}$ [20].
- R denotes the output of the SA if the final operation in S is a read operation. Hence, $\mathrm{R} \in \{0, 1, ?\}$, where ? denotes a random read output where the SA cannot reliably distinguish between 1 and 0. If the last operation was a write operation, then $R = -$.

Using this notation, it is possible to define the complete fault space, e.g., for single cell faults. For a single operation, there are 52 possible different FPs [20].

b) Address Decoders: AD faults (ADFs) in a single decoder are well studied and can be classified as static or dynamic faults [24]. There are four static ADFs: no access, multiple cells, multiple addresses, and other cells. These faults always lead to errors. Dynamic ADFs affect the timing of the AD and can be classified as Activation Delay Fault (ActD) and as Deactivation Delay Fault (DeactD) [25]. They do not always lead to errors.

c) Sense Amplifiers: SA faults (SAFs) can also be static and dynamic [26]. Static SAFs always cause the SA to output a fixed value, irrespective of the memory cell's contents. Dynamic SAFs are caused by slow SAs that fail to switch to the correct output in the allotted time. These faults do not always cause errors.

2) Fault Space Validation: Next, we validate the fault space by including the defect models from the previous step in a netlist and simulating it using the simulation set-up from [5], with the addition of the IUSF model from [9].

a) Memory Array: In [6], it was shown that 17/52 FPs are sensitized by using the traditional linear resistor defect models and the forming defect model. From [9], it follows that the IUSF can be described as $\langle 0\mathrm{w}1/\mathrm{U}/- \rangle$. This FP can be also sensitized by the forming defect [6]. However, the IUSF is intermittent in nature, and therefore it is not known when the fault will occur. Next, we determine the detectability of the faults. A fault is *easy-to-detect* (EtD) if its detection can be guaranteed by regular memory operations; e.g., $\langle 1\mathrm{w}0/1/- \rangle$, or $\langle 1\mathrm{r}1/\mathrm{U}/0 \rangle$. A fault is *hard-to-detect* (HtD) if its detection cannot be guaranteed by regular read/write operations; e.g., $\langle 0\mathrm{r}0/0/? \rangle$, or $\langle 0\mathrm{w}1/\mathrm{U}/- \rangle$. The IUSF is therefore a HtD fault whose detection requires extra effort.

b) Address Decoders and Sense Amplifiers: In [5], we have shown that static and dynamic ADFs and SAFs, can take place in RRAM-based CIM making use of Scouting logic.

B. Computation Configuration

Next, we perform fault modeling and analysis for the CC.

1) Fault Space Definition: In the CC, faults can occur in any CIM component contributing to such configuration, including the memory array, the ADs, and the SAs. Next, we define the fault space for each of these components.

a) Memory Array: A Scouting logic operation can be seen as a parallel read operation on two cells (operands) at once by a single (customized) SA, similar to how a two port-memory operates [27]. Hence, the $\langle \mathrm{S/F/R} \rangle$ notation needs to be extended to $\langle \mathrm{S}_1 : \mathrm{S}_2 \;/\; \mathrm{F}_1 : \mathrm{F}_2 \;/\; \mathrm{R} \rangle_{\mathrm{OP}}$ [6]. Here, ':' indicates that two operations happen in parallel, where $_1$ and $_2$ indicate which two cells are involved, and $_{\mathrm{OP}}$ indicates which logic operation; i.e., AND or OR is performed. For example, the FP $\langle 0\mathrm{r}0 : 0\mathrm{r}0/0 : 0/1 \rangle_{\mathrm{AND}}$ denotes a fault where performing an AND operation on two cells (operands) that store a '0' results in the wrong output '1'.

b) Address Decoders: When performing Scouting logic, the cells/operands have to be selected simultaneously. Hence, the row decoders of the CIM device have to operate concurrently; this is similar to what takes place in a memory with two read ports. Faults in such decoders are discussed in detail and reported in [28]; e.g., port interference can take place where one row decoder (for one operand) can erroneously select an additional row (additional operand) when two rows (operands) are simultaneously selected during the CIM operation.

c) Sense Amplifiers: SAs in the CC suffer from the faults as in the MC, except that the faults can now occur for every logic operation, i.e., for the AND reference and for the OR reference. For example, there can be dynamic slow SAF for the AND reference, and not for the OR reference.

2) Fault Space Validation: Next, a brief overview is given on some validation work done for defining the fault space.

a) Memory Array: A detailed validation of the CC fault space for memory array is reported in [6]; it considered both resistive defects as well as forming defects. Although this is far from being a complete analysis (e.g., not all unique RRAM defects are considered), it reveals quite interesting observations:

- There are defects that sensitize faults in the CC but not in the MC. Testing a CIM circuit only in the MC is therefore not sufficient.
- Defects can cause HtD faults in the MC, but EtD faults in the CC and vice versa. Hence, both configurations need to be tested.
- Many defects sensitize faults both in the MC and the CC.

It is worth noting that the unique IUSF can only manifest itself after a 0w1 operation; hence it is assumed that it can be only sensitized in the MC and not in the CC.

b) Address Decoders Sense Amplifiers: The work reported in [5] shows that static and dynamic ADFs and SAF can take place in CIM chips making use of Scouting logic.

VI. Device-Aware Test Development

Next, we show how the use the results from fault modeling and analysis can be used for test development.

A. Memory Configuration

The faults targeted in this paper for MC consist of static single memory cell array faults (MCAFs), static and dynamic address decoders faults (ADFs), as well as static and dynamic sense amplifier faults (SAFs).

To detect *static* faults targeted in this work, one can develop a march test solution. March test algorithms have been very popular memory test solutions since the 1980's [29]. The following march test was presented in [6] for the detection of faults in MC of scouting logic-based CIM; these consist of static EtD MCAFs, static ADFs, and static SAFs:

$$\textbf{March-EtD-MC} = \{\Updownarrow(\mathrm{w}0)\,;\Uparrow(\mathrm{r}0,\mathrm{w}1)\,;\Downarrow(\mathrm{r}1,\mathrm{w}0)\,;\Updownarrow(\mathrm{r}0)\}\,.$$

Here, $\Updownarrow$ indicates any addressing order, and $\Uparrow$ ($\Downarrow$) indicates increasing (decreasing) addressing order. As already mentioned before, typical march tests cannot guarantee the detection of HtD faults; hence additional effort is needed. One option is to augment the previous march test with a special weak write operation ($\hat{\mathrm{w}}$) supported through DFT as reported in [30], resulting in the following:

$$\textbf{March-HtD-MC} = \\ \{\Updownarrow(\mathrm{w}0)\,;\Downarrow(\mathrm{r}0,\mathrm{w}1,\hat{\mathrm{w}}0)\,;\Uparrow(\mathrm{r}1,\mathrm{w}0)\,;\Updownarrow(\hat{\mathrm{w}}1,\mathrm{r}0)\}\,.$$

Note the $\hat{\mathrm{w}}$ will not bring any change in the state for a healthy/good memory cell, while this will change the state of the weak/bad memory cell.

Covering dynamic ADFs and SAFs considered in this work put additional requirements on the table. For dynamic ADFs, special address transitioning orders need to be adapted; e.g., using addresses that have a hamming distance of 1 between them [25]. For the detection of the dynamic SAFs, back-to-back operations that quickly flip the SA's output can be added to the above algorithms [26].

Obviously, the above test cannot guarantee the detection of the IUSF (being an HtD MCAF) due to its intermittent nature [9]. This fault can only be probabilistically detected using, for example, the following March algorithm:

$$\textbf{March-IUSF-MC} = \left\{\Updownarrow(\mathrm{w}0,\mathrm{w}1,\mathrm{r}1)^k\right\},$$

where k indicates the number of times the sequence is applied. If we assume that reading a cell in 'U' state results the same probability of getting '1' or '0' (i.e., 50 %, due to process variations in the SA and the reference circuit), and that the occurrence probability of IUSF is P_{IUSF}, then the detection probability is: $P_{\mathrm{d}}=1-(1-(P_{\mathrm{IUSF}}\cdot 50\,\%))^k$. Assuming that P_{IUSF}=1.068 % results in $k=560$ to realize a fault coverage (FC) of FC=95 %, and in k=1291 to realize FC=99 % [9]. Hence, realizing high FC needs a long test time; not to mention the potential impact of repeating memory accesses on the cell endurance. Hence, dedicated DFTs are needed.

B. Computation Configuration

In [6], two tests for the CC are presented as well for both EtD and HtD faults. The following march algorithm can detect all EtD faults in the memory array in the CC:

$$\textbf{March-EtD-CC} = \{\Updownarrow (\mathrm{w0})\,; \Uparrow (\mathrm{r0}, \mathrm{w1})\,; \Downarrow (\mathrm{r1}, \mathrm{w0})\,; \\ \Updownarrow (\mathrm{w1}_r : \mathrm{n}, \mathrm{r1}_r : \mathrm{r0}_{r+1}, \mathrm{r0}_{r+1} : \mathrm{r1}_r, \mathrm{r1}_r : \mathrm{r0}_{r+1}, \\ \mathrm{w0}_r : \mathrm{n})_{AND,AND,OR}\Big\}\,,$$

where n denotes no operation. Note that the computing operation accesses two cells in the same column *simultaneously* (i.e., rows r and $r+1$). Again using the DFT from [30], the following march algorithm can detect the HtD faults in the memory array, except for the IUSF:

$$\textbf{March-HtD-CC} = \{\Uparrow (\mathrm{w1})\,; \Uparrow (\mathrm{r1}, \hat{\mathrm{w}}1, \mathrm{r1})\,; \\ \Downarrow (\mathrm{w0}, \hat{\mathrm{w}}1)\,; \Downarrow (\mathrm{r0}_r : \mathrm{n}, \mathrm{w1}_r : \mathrm{n}, \\ \mathrm{r1}_r : \mathrm{r0}_{r+1}, \mathrm{r0}_{r+1} : \mathrm{r1}_r, \mathrm{w0}_r : \mathrm{n})_{R,OR,AND}\Big\}\,.$$

This test also cannot guarantee the detection of the IUSF. However, it is possible to use the compute operations to increase the detection probabilities of the IUSF. From Fig. 3, it follows that the equivalent resistance of a '1' in parallel with a cell in 'U' is higher than the reference for the AND operation. Thus, this operation can be used to detect cells in 'U', resulting in the following probabilistic march algorithm:

$$\textbf{March-IUSF-CC} = \{\Updownarrow (\mathrm{n}_r : \mathrm{w1}_{r+1})\,; \\ \Updownarrow (\mathrm{w0}_r : \mathrm{n}_{r+1}, \mathrm{w1}_r : \mathrm{n}_{r+1}, \mathrm{r1}_r : \mathrm{r1}_{r+1})^k_{AND}\Big\}\,.$$

The detection capabilities of March-IUSF-CC then change to: P_d=$1-(1-P_\mathrm{IUSF})^k$, i.e., $k = 279$ results in FC=95 %, and $k = 429$ results in FC=99%, which is significantly higher than that of March-IUSF-MC.

VII. Discussion and Conclusion

This paper introduces a structured test development approach for CIM architectures, and demonstrates it on a RRAM-based implementation of Scouting logic. Some key conclusions:

Usage of CIM to detect memory faults: We have demonstrated that CIM capabilities can be used to detect cells that are in the 'U' state (for example due to an IUSF). It enables the detection of faulty cells in the 'L' or 'H' state. For example, the equivalent resistance of a cell in 'H' and '0' may be lower than the AND reference; the fault can than be detected by performing an AND operation.

Unique faults: Testing the CIM die in the MC only is not enough. CIM in CC gives rise to new and unique faults because the CC makes partial use of the MC hardware, but also of unique components, e.g., simultaneous usage of ADs. Hence, dedicated tests for this configuration need to be used. Further, some defects sensitize EtD faults in the MC and HtD faults in the CC. Hence, it is possible to optimize the test by selecting the EtD faults for each configuration.

Generality of the Proposed Approach: The core of the presented test development approach is to identify which components have been modified to allow for CIM and to subsequently develop test to detect faults in them. We have demonstrated that approach for RRAM-based Scouting logic, but it also applies to other CIM schemes, such as vector-matrix-multiplication architectures [31]. For example, in such an architecture, the ADs are modified, and the SAs are replaced with analog-to-digital converters. A structured test should be able to detect faults in these modified circuits.

References

[1] D. A. Patterson, "Future of Computer Architecture," in *BEARS*, 2006.
[2] L. Xie *et al.*, "Scouting Logic: A Novel Memristor-Based Logic Design for Resistive Computing," in *ISVLSI*, 2017.
[3] S. Li *et al.*, "Pinatubo: a processing-in-memory architecture for bulk bitwise operations in emerging non-volatile memories," in *DAC*, New York, New York, USA, 2016.
[4] T.-L. Tsai *et al.*, "Testing of In-Memory-Computing 8T SRAMs," in *DFT*, 2019.
[5] S. Hamdioui *et al.*, "Testing Computation-in-Memory Architectures Based on Emerging Memories," in *ITC*, 2019.
[6] M. Fieback *et al.*, "Testing Scouting Logic-Based Computation-in-Memory Architectures," in *ETS*, 2020.
[7] E. I. Vatajelu *et al.*, "Challenges and Solutions in Emerging Memory Testing," *IEEE TETC*, 2017.
[8] L. Wu *et al.*, "Defect and Fault Modeling Framework for STT-MRAM Testing," *IEEE TETC*, 2019.
[9] M. Fieback *et al.*, "Intermittent Undefined State Fault in RRAMs," in *ETS*, 2021.
[10] J. F. Li *et al.*, "Testing of Configurable 8T SRAMs for In-Memory Computing," in *Proceedings of the Asian Test Symposium*, 2020.
[11] T. L. Tsai *et al.*, "Testing of in-memory-computing memories with 8 T SRAMs," *MR*, 2021.
[12] L. Ammoura *et al.*, "Preliminary Defect Analysis of 8T SRAM Cells for In-Memory Computing Architectures," in *DTIS*, 2021.
[13] S. M. Nair *et al.*, "Defect Characterization and Test Generation for Spintronic-based Compute-In-Memory," in *ETS*, 2020.
[14] C. Munch *et al.*, "Testing Resistive Memory based Neuromorphic Architectures using Reference Trimming," in *DATE*, 2021.
[15] H. A. D. Nguyen *et al.*, "A Classification of Memory-Centric Computing," *ACM JETC*, 2020.
[16] M. A. Lebdeh *et al.*, "Memristive Device Based Circuits for Computation-in-Memory Architectures," in *ISCAS*, 2019.
[17] J. Yu *et al.*, "Memristive devices for computation-in-memory," in *DATE*, 2018.
[18] M. Fieback *et al.*, "Device-Aware Test: A New test Approach Towards DPPB Level," in *ITC*, 2019.
[19] M. Taouil *et al.*, *Patent NL2023751B1 Device-Aware Test for Memory Units*, 2020.
[20] M. Fieback *et al.*, "Defects, Fault Modeling, and Test Development Framework for RRAMs," *ACM JETC*, 2022.
[21] L. Wu *et al.*, "Pinhole Defect Characterization and Fault Modeling for STT-MRAM Testing," in *ETS*, 2019.
[22] M. Syrzycki, "Modeling of Gate Oxide Shorts in MOS Transistors," *IEEE TCADICS*, 1989.
[23] S. Hamdioui *et al.*, "Testing static and dynamic faults in random access memories," in *VTS*, 2002.
[24] A. J. van de Goor, *Testing Semiconductor Memories - Theory and Practice*. 1991.
[25] S. Hamdioui *et al.*, "Opens and Delay Faults in CMOS RAM Address Decoders," *IEEE TC*, 2006.
[26] A. van de Goor *et al.*, "Detecting faults in the peripheral circuits and an evaluation of SRAM tests," in *ICT*, 2004.
[27] S. Hamdioui *et al.*, "Efficient tests for realistic faults in dual-port SRAMs," *IEEE TC*, 2002.
[28] S. Hamdioui *et al.*, "Address decoder faults and their tests for two-port memories," in *MTDT*, 1998.
[29] M. S. Abadir *et al.*, "Functional Testing of Semiconductor Random Access Memories," *CS*, 1983.
[30] S. Hamdioui *et al.*, "Testing Open Defects in Memristor-Based Memories," *IEEE TC*, 2015.
[31] A. Velasquez *et al.*, "Parallel boolean matrix multiplication in linear time using rectifying memristors," in *ISCAS*, 2016.