2024 Design, Automation & Test in Europe Conference (DATE 2024)

A Novel March Test Algorithm for Testing 8T SRAM-based IMC Architectures

Lila Ammoura [1] Marie-Lise Flottes [1] Patrick Girard [1] Jean-Philippe Noel [2] Arnaud Virazel [1]

[1] *LIRMM – Univ. of Montpellier / CNRS*
F-34392 Montpellier, France
<name>@lirmm.fr

[2] *Univ. Grenoble Alpes, CEA, LIST*
F-38000 Grenoble, France
jean-philippe.noel@cea.fr

***Abstract*—The shift towards data-centric computing paradigms has given rise to new architectural approaches aimed at minimizing data movement and enhancing computational efficiency. In this context, In-Memory Computing (IMC) architectures have gained prominence for their ability to perform processing tasks within the memory array, reducing the recourse to data transfers. However, the susceptibility of these new paradigms to manufacturing defects poses a critical test challenge. This paper presents a novel March-like test algorithm for 8T SRAM-based IMC architectures, addressing the imperative need for comprehensive read port related defect coverage. The proposed algorithm achieves complete coverage of potential read port defects while maintaining the level of complexity equivalent to existing state-of-the-art test solutions.**

Keywords— In-Memory Computing, 8T SRAM bitcell, Testing, March test algorithm, DfT.

I. Introduction

The field of high-performance computing is undergoing a transformative evolution with the emergence of a new conceptual framework. This framework calls for a move away from the conventional compute-centric model towards an innovative data-centric approach [1]. At its core, this new methodology seeks to optimize the design of computer systems by minimizing data transfers. This optimization is achieved by strategically executing processing tasks close to data storage locations inside the memory architecture of the system [2]. The primary objective of this paradigm shift is to enhance computational efficiency by reducing the latency and energy consumption associated with frequent data transfers [3].

Within this evolving paradigm, a range of alternative architectural concepts have been investigated. Notable among these is the "Near-memory" concept, which aims to execute data processing operations in immediate proximity to dedicated storage nodes [4]. Furthermore, a particularly promising strategy to address the challenges posed by data-intensive applications involves the adoption of In-Memory Computing (IMC) architectures [5-9]. These architectures promote the execution of processing tasks within the memory array itself, thereby circumventing the necessity for extensive data transfers. By intimately interweaving processing and memory functionalities, IMC architectures strive to surmount the limitations that have constrained conventional computation-centric models.

However, a critical aspect requiring particular attention in the exploration of these pioneering paradigms concerns the susceptibility of these systems to manufacturing defects [10]. Similarly to conventional memories designed using identical technologies, these innovative computing approaches are not immune to the imperfections that can arise during manufacturing processes [11]. As these cutting-edge paradigms move on to practical implementation in contemporary data processing units, it is imperative to develop specific effective test solutions. These solutions will be crucial in solving the specific challenges associated with integrating logic and processing components directly into the memory part of the IMC architecture. Through the development of specialized test methodologies, these solutions will effectively address the increased complexities of IMC-based architectures and contribute to guaranteeing their reliability.

To the best of our knowledge, there exists only one paper that proposes a March-like algorithm for testing 8T SRAM-based IMC architectures [12]. In this paper, the authors demonstrate the need to test IMC architectures in both their memory and computation configurations and propose a March-like test algorithm. However, it's worth noting that this previously proposed testing solution primarily focuses on testing typical functional faults, without addressing the structural aspect of the IMC architecture. In fact, recent works have shown that this proposed test solution does not encompass all the possible defects that can affect the read port of the IMC 8T SRAM bitcell [13-15]. Consequently, the analysis of resistive-short and resistive-open defects injected into the read port was conducted in [15]. Results of this analysis revealed that performing global computation on all bitcells within the same column enhances the detectability of specific defects.

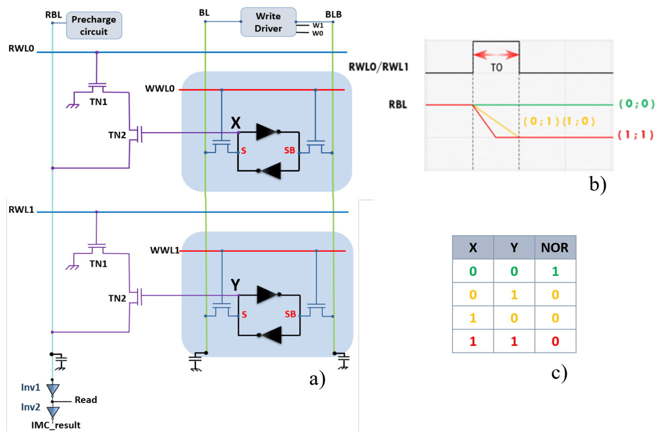
In this paper, we develop a novel March test algorithm for testing 8T SRAM-based IMC architectures. The proposed March test solution is rooted in the insights gained from defect modeling and fault modeling research reported in [15]. Notably, our algorithm achieves a complete coverage of potential read port defects, even in their worst-case scenarios, all while maintaining an equivalent level of complexity compared to state-of-the-art test solutions. The execution of our proposed test algorithm requires a specific global access per memory column. Therefore, a Design for Testability (DfT) solution is mandatory to adapt the function of the address decoder. The DfT requirements are discussed at the end of the paper.

The remainder of this paper is structured as follows. Section II provides a comprehensive overview of the structural aspects and the operating of an 8T SRAM bitcell in both memory and computing modes. Section III presents background information on existing test solutions and reports the latest defect modeling and fault modeling results, which serve as the foundation for developing the dedicated March test algorithm. Section IV presents the different development steps of the novel March-like test algorithm for 8T SRAM-based IMC architectures and discusses the DfT requirement that must be embedded in the address decoder. Finally, Section V concludes the paper and gives future perspectives.

II. 8T SRAM-Based IMC Overview

A. *IMC 8T SRAM bitcells principle*

IMC architectures enable the execution of computations directly within the memory, reducing the need to transfer data to an external computing node. These architectures can function in two distinct modes: memory mode and computing mode. Within the memory mode, the memory carries out read or write operations on a targeted word. Conversely, in the computing mode, the memory undertakes operations involving a minimum of two addressed words.



X	Y	NOR
0	0	1
0	1	0
1	0	0
1	1	0

Fig. 1. a) An example of 8T SRAM bitcells for in-memory computing, b) a waveform showing the execution of a NOR operation and c) the resulting truth table on IMC_result output.

1) *Memory mode*

a) *Write operation*

Under the memory mode, the process of writing in an 8T SRAM bitcell operates in two sequential steps:

- Step 1: In the initial step, as depicted in Fig. 1.a, the write driver sets the bit lines to the desired values by configuring the Bit Line (BL) to carry a specific state and its complementary state on the Bit Line Bar (BLB).
- Step 2: Subsequently, the address decoder activates the appropriate Write Word Line (WWL) to a high state. Given the significant size disparity between the access transistors and the bitcell inverters, the internal signals within the bitcell are compelled to align with the values present on the bit lines. This compels the bistable circuit to transition into a new stable configuration.

b) *Read operation*

For the purpose of reading the contents of the 8T SRAM bitcell, the Read Bit Line (RBL), initially charged to Vdd, starts the operation with a floating ‘1’. Subsequently, the activation of the Read Word Line (RWL) takes place. Let us examine the following two scenarios:

- Scenario 1 - The read bitcell stores a logical ‘0’ (S = ‘0’): The NMOS transistor TN2 within the read port remains in an off-state. This leads to maintain the RBL at Vdd, resulting in the observation of a logical ‘0’ at the read output port (accounting for the presence of an inverter at the “Read” output of RBL).
- Scenario 2 – The read bitcell stores a logical ‘1’ (S = ‘1’): The discharge of RBL occurs through TN1 and TN2. Consequently, after the output inverter, a logic ‘1’ is observed on the Read output port.

2) *Computing mode*

The computing mode involves conducting a read operation concurrently among a minimum of two (or more) 8T SRAM bitcells by simultaneously activating their respective RWL signals. This leads to the output of the read port exhibiting a NOR behavior from the selected 8T SRAM bitcells.

Let us detail the computing mode by considering Fig. 1. Suppose RWL0 and RWL1, aligned with operand X and operand Y respectively, are concurrently activated, as depicted in Fig. 1.a. The precharged RBL remains at its Vdd state solely when both operands X and Y (i.e., logic content of each bitcell) are in logic ‘0’. In essence, as illustrated in Fig. 1.b, by jointly triggering RWL0 and RWL1 signals for a duration of T0, the RBL retains a high state only when both X = ‘0’ and Y = ‘0’, and it fully discharges as soon as at least one of the operands is in logic state ‘1’. This process computes the NOR operation of both operands X and Y, and the result is inferred from the voltage level of the RBL.

An inverter (Inv1) is connected to the RBL, causing the output of the inverter to drop low if the RBL remains high. Consequently, the output IMC_result of the successive inverter (Inv2) exhibits a NOR behavior (the truth table is outlined in Fig. 1.c). It is noteworthy that when the complementary state is stored within the bitcells (i.e., node SB holds the input data for computation), the output IMC_result of the successive inverter (Inv2) demonstrates an AND behavior.

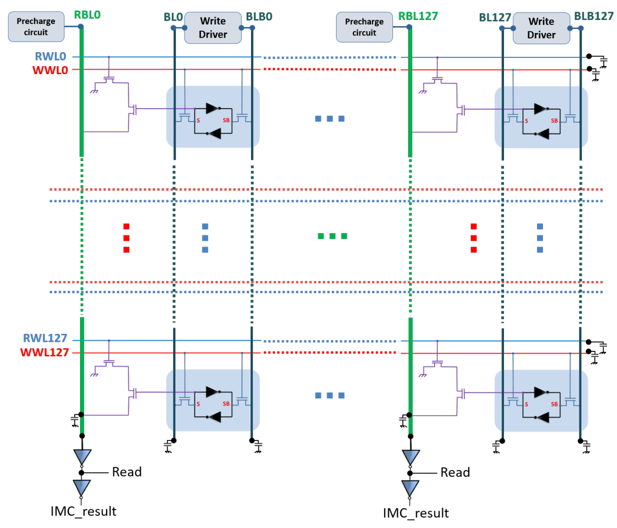


Fig. 2. Considered 128x128 matrix model with layout extraction of parasitic capacities.

B. *Considered IMC SRAM array*

Our study aimed to understand the electrical behavior in real-world scenarios. To achieve this, we employed a model, illustrated in Fig. 2, representing a 128x128 bitcell array developed using the 28 nm FD-SOI process technology. Within this model, we integrated write drivers designed to manage writing operations within bitcells. Additionally, precharge circuits were incorporated to sustain the RBL signals at Vdd levels, a prerequisite for executing read operations and array-level computations. Further enhancing the authenticity of our model, we conducted a layout extraction process targeting the parasitic capacitances of key signals (such as BL/BLB, RBL, and RWL). This meticulous inclusion contributes to results that closely emulate those attainable in an actual silicon implementation.

III. Background on 8T SRAM-based IMC Testing

A. Existing 8T SRAM Test solutions

1) Memory mode testing

In memory mode, IMC architecture performs read/write operations, mirroring the behavior of conventional memory. This operational resemblance allows for IMC memory to undergo testing using identical methods as those employed for conventional memory testing [12, 16, 17].

The identification and assessment of defects in memory chips typically involve representing them as functional faults, with their detection achieved through the application of functional tests based on specific Functional Fault Models (FFMs) [18]. A systematic approach is crucial for enhancing the performance and reliability of memory devices. The typical test development process for SRAMs encompasses three primary stages:

- *Defect Analysis*: This phase involves utilizing a physical model of the memory, where defect injection campaigns are conducted to analyze and characterize defects.
- *Fault Modeling*: During this stage, the objective is to identify an appropriate FFM for each type of analyzed defect.
- *Test Algorithm Development*: This step entails the development of test algorithms, such as March-like tests, designed to encompass all potential FFMs encountered within a particular memory technology [19].

In [12], the authors consider the well-known March C- as test algorithm of IMC 8T SRAM in memory mode. This algorithm presents a sequence of March Elements (ME). Each ME is a sequence of memory operations, denoted as:

$$\text{March C-} = \{\Updownarrow(w0); \Uparrow(r0, w1); \Uparrow(r1, w0); \Downarrow(r0, w1); \Downarrow(r1, w0); \Updownarrow(r0)\}.$$

March C- possesses a complexity proportional to 10N, where N represents the number of memory bitcells. It ensures comprehensive coverage of various static FFM, including Stuck-At-Faults (SAF), Transition Faults (TF), idempotent and inversion Coupling Faults (CFid, CFin, respectively), and Address decoder Faults (AF).

However, in [13-15], it is demonstrated that March C- operations does not encompass detection of all potential resistive defects located within the read port of the IMC 8T SRAM bitcell. Consequently, it is imperative to consider all defects located in the isolated read port during the generation process of the test algorithm.

2) Computing mode testing

In [12], a March-like test algorithm, denoted as March C-8, was introduced specifically for testing 8T SRAM-based IMC architectures in computing mode. Notably, the IMC architecture employed in [12] is configured to execute NOR, NAND and XOR operations during computing mode, by deploying two threshold inverters positioned at the output of the read port. So, to test computing operations, two specific requirements are considered:

1. Requirement #1 entails executing either a read operation or a NOR operation, in order to identify faults stemming from excessive leakage current when all data bits are ‘0’.
2. Requirement #2 involves the execution of NAND operations, particularly in scenarios with operands (0,1) or (1,0), to address worst-case situations.

The initial requirement aligns with the first two operations inherent in the March C- test algorithm. To fulfill the second requirement, March C- algorithm has been extended by incorporating two additional NAND operations between the cell currently being processed i and the next one i+1 in the 2nd and 4th ME, thereby becoming March C-8, denoted as:

$$\text{March C-8} = \{\Updownarrow(w0); \Uparrow(r0, w1, \mathbf{NAND}_i^{i+1}); \Uparrow(r1, w0); \Downarrow(r0, w1, \mathbf{NAND}_i^{i-1}); \Downarrow(r1, w0); \Updownarrow(r0)\}.$$

The proposed March C-8 algorithm primarily focuses on functional testing, ensuring that the IMC architecture functions correctly under normal operational conditions in computing mode. However, it falls short in addressing the critical aspect of structural testing, particularly concerning defects that could potentially impact the read port of the IMC 8T SRAM bitcells, as demonstrated in [15]. While functional testing is essential for verifying the intended operation of the IMC architecture, structural testing is crucial for identifying and diagnosing physical defects that might compromise its reliability [16]. Thus, it remains a significant gap in the testing approach, as it fails to comprehensively account for the potential defects that may affect the read port's integrity and overall performance, hence the need for a comprehensive testing approach. Therefore, the goal outlined in [15] aimed at systematically injecting and analyzing all resistive defects capable of impacting the read port. This objective served as the foundation for deducing fault models, that describe the faulty behavior in the presence of each injected defect. The following two subsections outline the structural approach adopted in [15] to analyze the resistive defects introduced at the read port. Additionally, it provides a short summary of the findings obtained from this analysis.

B. Defect modeling approach

The structural approach considered in [15] aims to assess the potential impact of each defect on read/write/computing operations on the defective bitcell as well as globally on the array. In the framework for injecting resistive-short and resistive-open defects (illustrated in Fig. 3), a monitoring bitcell denoted as (*i*; *j*) (indicating its position in row *i* and column *j*) was selected as the target for the injection of a single defect at its read port. The defect analysis was proceeded hierarchically as follows:

- *Stand Alone Analysis (SA_Analysis)*: This assesses the local impact of the defect on the defective bitcell itself during memory mode operations.
- *Neighborhood Analysis (N_Analysis)*: It consists of two stages: *i)* Evaluation of the impact on defect-free surrounding bitcells when performing memory mode operations on the defective bitcell. *ii)* Assessment of the localized impact on the defective bitcell during memory mode operations performed on defect-free surrounding bitcells.
- *Computation Analysis (C_Analysis)*: This comprises two phases:

i) Examination of the influence on computing mode operations between the defective (aggressor) bitcell and at least one defect-free (victim) bitcell in the same column (i.e., NOR(c_a;c_v)).

ii) Evaluation of the impact on computing mode operations between at least two victim bitcells located in the same column as the defective one (i.e., NOR(c_v;c_v)).

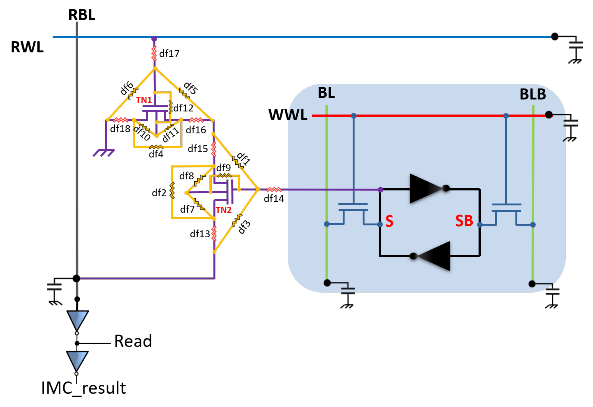


Fig. 3. Resistive-short & resistive-open defects injection in the read port of an 8T SRAM bitcell.

C. Fault modeling results

Throughout the structural analysis introduced in the previous subsection, the extraction of information concerning the influence of each defect on read/write/computing operations was derived. Subsequently, Hspice simulation were conducted to determine the critical resistances (i.e., R_c) at which the defects (shorts/opens) became detectable. Following this analysis, Fault Primitives (FP) for each defect were derived. As detailed in [19], an FP is represented as follows:

- <S/F/R> when a single bitcell is involved in sensitizing a fault where it appears. Here, "S" signifies the Sensitizing Operation Sequence (SOS) responsible for triggering the fault; S ∊ {0, 1, w0, w1, w↑, w↓, r0, r1}.
- <Sa,Sv/F/R> when two bitcells are engaged in sensitizing the fault. "Sa" describes the sensitizing operation or state of the aggressor bitcell, while "Sv" describes the sensitizing operation or state of the victim bitcell. Si ∊ {0, 1, X, w0, w1, w↑, w↓, r0, r1} (i∊{a, v}), where X is the don't care value X∊{0, 1}.

In both notations, "F" characterizes the value or behavior of the faulty bitcell, which can be any of {0, 1, ↑, ↓, -} where ↑ (resp. ↓) indicating that the faulty bitcell undergoes a transition. "R" pertains to the logic output level of a read operation when "S" includes read operations. Typically, it assumes one of the values {0, 1, -}, with '-' signifying that no read operation is required for the SOS.

For each injected defect, we have deduced the sensitization sequence required for its detection in the worst-case scenario. From these deduced sensitization sequences, we have gathered essential information on the optimal detectability conditions of each defect. The results obtained from the fault modeling process are summarized in Table 1. The first column addresses defects related to the read port (c.f. Fig. 3), while the second column presents the best-case fault primitives required for detecting these defects under their worst-case conditions. This implies that the execution of these fault primitives allows the detection of smaller resistive-open defects or, conversely, larger resistive-short defects.

From data reported in Table 1, we can identify two distinct case studies:

Case 1: There are defects, such as df3, df6, and df12, that can be sensitized in their worst-case conditions through the execution of a conventional read operation. To illustrate, let us consider the case of resistive-short defect df3 (as shown in the fourth row of Table 1). This defect can be detected with a straightforward read operation applied to the bitcell initially set to a logical '0'. If the execution of the r0 operation is incorrect, the read port will provide a logical '1' instead of logical '0'. Thus, the corresponding fault primitive is denoted as FP3: <0r0 /0/ 1> to describe this scenario.

Case 2: There are defects, such as df2 and df7, for which sensitization in their worst-case scenarios requires a computational operation involving two bitcells. On the other hand, certain defects present greater complexity in detection under worst-case conditions. For instance, df1, df9, df13-df18 necessitate a global NOR operation (i.e., a NOR operation involving all bitcell of the column), while the group consisting of df4, df5, df8, and df11 requires a global NOR operation with the exception of aggressor bitcell. To illustrate this case, let us consider the first group of defects reported in Table 1, including the resistive-short defects df1 and df9, as well as all resistive-open defects ranging from df13 to df18. Their best-case FP occurs when considering all bitcells within the same column for a NOR operation, with all the bitcells initialized to logical '0' except one bitcell initialized to logical '1'. So, the corresponding FP is denoted as FP1:

<1, 0^{Nc-1} NOR(1;0^{Nc-1}) /0^{Nc-1}/1>

where Nc represents to the total number of bitcells within the column. Further details of this FP are elaborated below. A logic '1' is initially stored in the defective bitcell. A logic '0' is initially stored in Nc-1 bitcells of the same column as the defective one. Then, a NOR(1;0^{Nc-1}) operation is performed on Nc selected bitcells. The Nc-1 bitcells remain at logic '0'. The output level of the logical operation is a logic '1'.

TABLE I. SUMMARY OF THE FAULT MODELING RESULTS

Defects	FP: <S/F/R> / <Sa, Sv/F/R>
df1, df9, df13-df18	FP1: <1, 0^{Nc-1} NOR (1;0^{Nc-1}) /0^{Nc-1}/ 1>
df2, df7	FP2: <0, 0^1 NOR (0;0^1) /0^1/ 1>
df3	FP3: <0r0 /0/ 1>
df4, df5, df8, df11	FP4: <1, 0^{Nc-1} NOR (0;0^{Nc-2}) /0^{Nc-1}/ 1>
df6, df12	FP5: <1, 1r1 /1/ 0>

IV. Test Development

A. Principle

Development of the dedicated test algorithm is based on the integration of insights obtained through defect analysis and fault modeling. These insights provide us with crucial information concerning the optimal conditions for effective defect detections. This information is subsequently translated into a sequence of memory and computing operations, serving as the foundation of our test algorithm. By executing this algorithm, we will systematically assess the functionality of the IMC memory, ensuring its reliable operation in computing mode, and promptly detecting any potential defect.

B. Proposed Test Algorithms

The process of developing the dedicated algorithm involves the steps where we translate the FPs into a sequence of memory and computing operations as well. This transformation effectively converts the FP into a structured March-like test, optimized for covering potential defects within the read port of IMC 8T SRAM bitcell. This March, named March 8T-Read Port Defect (RPD), is denoted as follows:

March 8T-RPD = {⇕ (w0); ⇑ (r0, w1, $\mathrm{NOR}(0; 0^{Nc-2})$, w0);
⇓($\mathrm{NOR}(0; 0^1)_j^i$,w1, $\mathrm{NOR}(1; 0^{Nc-1})$, w0)}.
with i = 1; then i = i+2 & j = 2; then j = j+2.

March 8T-RPD test algorithm comprises three key MEs:

- The first element, a w0 operation, aims to initialize the entire memory array to the '0' state.
- Within the second element, we first act a r0 operation, which already includes the FP3 (see Table 1). Subsequently, a w1 operation is added to change the state of a bitcell to '1'. This operation satisfies the integration of the FP4 by inserting the $\mathrm{NOR}(0; 0^{Nc-2})$ computing operation. To conclude this element, another operation, w0, is employed to reset the bitcell state back to '0' before selecting the next address. This second Mach element embedded FP1 and FP4.
- In the last element, a $\mathrm{NOR}(0; 0^1)_j^i$ computing operation is introduced, performed between two bitcells, namely $cell_i$ and $cell_j$, initially with i = 1 and j=2. This computing operation is conducted iteratively on two different bitcells by adjusting the step to 2. It satisfies the FP2. Subsequently, a W1 operation is added to change the state of one bitcell, enabling the integration of the global $\mathrm{NOR}(1; 0^{Nc-1})$ operation (i.e., FP1). Finally, a W0 operation is utilized to restore the bitcell state to '0'. This last March element embedded FP2 and FP1.

Conventional March algorithms are designed for comprehensive testing of the memory array and its periphery (e.g., address decoder). They primary focus on detecting static faults in memory bitcells (e.g., stuck-at faults) and double-cell faults such as coupling faults (e.g., CFin/CFid). The March C- test algorithm is widely used thanks to its fault-coverage capabilities. In assessing the effectiveness of the March 8T-RPD algorithm, it is noteworthy that this algorithm delivers complete coverage for Static Faults including SAF, AF, TF, CFin (with the exception of CFid), and all read port-related defects within 8T SRAM-based IMC architectures.

The possibility of merging dedicated March 8T-IMC with March C- test algorithm offers an opportunity to combine the specialized detection capabilities of the dedicated March 8T-RPD with the extensive static fault coverage provided by March C-. This integration results in a novel test algorithm, named March IMC-8T. To achieve this, we developed March IMC-8T through the following steps:

1) The initial March Element (i.e., ME0) of March C- remains unchanged.
2) As previously explained in section II.2, a computing operation is equivalent to read simultaneously two bitcells (at least) within the same column. Therefore, we replaced the r0 operation in the second ME1 of March C- with a $\mathbf{NOR(0; 0^1)_j^i}$ operation performed between $cell_i$ and $cell_j$, initially with i = 1 and j = 2. This NOR operation is applied iteratively to different bitcell pairs, with the step of 2. This modification saves time by eliminating redundancy, and achieves 50% less execution time.
3) Elements ME2, ME3, and ME4 remain unchanged as they are responsible for testing TFs, AFs, and CFin/CFid static faults.
4) In ME5, r0 is replaced by a global NOR operation, which is performed only once per column. The address symbol (i.e., ⇕) is replaced by "|" to indicate the once-per-column operation. This effectively reduces test time by dividing it over Nc.

At this point, the fault detection capabilities of March C- are not modified. Nevertheless, the modifications from memory operation to computing operations allow us to integrate part of the test conditions related to the read port defects (FP2, FP3 and FP5) and optimize the test time.

5) Lastly, a new element, ME6, is introduced. ME6 serves as an integration point for incorporating defects detection conditions specific to FP1 and FP4**.**

The dedicated March IMC-8T test algorithm is outlined below. Its complexity can be expressed as follows:

$$C = (12 + \tfrac{1}{2} + \tfrac{1}{Nc})\, N \approx 12N.$$

March IMC-8T = {
ME0: ⇕ (w0);
ME1: ⇑ ($\mathrm{NOR}(0; 0^1)_j^i$, w1);
ME2: ⇑ (r1, w0);
ME3: ⇓ (r0, w1);
ME4: ⇓ (r1, w0);
ME5: | ($\mathrm{NOR}(0; 0^{Nc-1})$);
ME6: ⇓ (w1, $\mathrm{NOR}(1; 0^{Nc-1})$, $\mathrm{NOR}(0; 0^{Nc-2})$,w0)
}; with i = 1; then i = i+2 & j = 2; then j = j+2.

C. Comparison

In the context of 8T SRAM-based IMC architectures testing, we compare our two dedicated March-like algorithms, i.e., March 8T-RPD and March IMC-8T, with March C-8 [12], as well as the widely adopted March C- test algorithm. Table 2 provides a comparison that highlights the static faults coverage (SAF, AF, TF, CFin/CFid, respectively), Read Port Defects (RPD) coverage, and also the coverage in their worst-case scenario (see the penultimate column). The last column shows the complexity of the quoted algorithms.

March IMC-8T and the proposed March C-8 test represent two distinct approaches in the development of test algorithms for 8T SRAM-based IMC architectures. Notably, March C-8 primarily concentrates on testing typical functional faults without addressing the structural aspect of the IMC architecture. With a complexity of 12N, March C-8 shares the same fault coverage as March C-. It falls short in handling read port defects, achieving only 77.8% coverage. Furthermore, it tests only 16.7% of read port defects under their critical conditions. In contrast, our dedicated March IMC-8T algorithm, despite sharing an equivalent complexity of 12N, distinguishes itself through its comprehensive testing approach. It integrates the detection of static faults and achieves full coverage of read port related defects, even under worst-case conditions.

TABLE II. ALGORITHM COMPARISON: STATIC FAULT AND READ PORT DEFECT COVERAGE FOR 8T SRAM-BASED IMC ARCHITECTURES

March Algorithm	SAF %	AF %	TF %	CFin CFid %	RPD (Worst-Case) %	C
C-	100	100	100	100	77.8 (16.7)	10N
C-8 [12]	100	100	100	100	77.8 (16.7)	12N
8T-RPD	100	100	100	50	100 (100)	~8N
IMC-8T	100	100	100	100	100 (100)	~12N

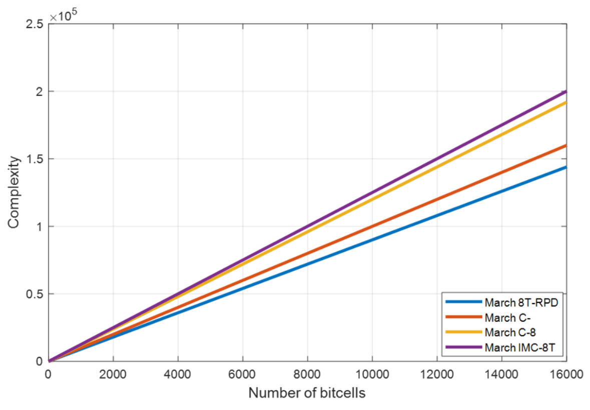


Fig. 4. The complexity variation of the quoted March test algorithms based on total memory bitcells.

D. *Required DfT*

The dedicated March IMC-8T algorithm includes conventional memory operations as well as computing operations (i.e., NOR operations). These computing operations involve either two bitcells or the entire column (i.e., ME5, ME6), while others exclude a single bitcell from the column (i.e., ME6). Therefore, to implement these computing operations, two new functionalities must be integrated into the address decoder: one functionality to ensure the simultaneous activation of all RWLs, and a second one to activate all RWLs except one. To achieve this, a crucial prerequisite is the incorporation of a Design for Testability (DfT) approach to customize the functionality of the address decoder to enable the application of March IMC-8T.

Typically, a conventional address decoder can access one address at once. However, in the context of IMC, a dedicated address decoder has been proposed in [20]. It allows the simultaneous activation of two address rows at once by exploiting two different row decoders and registers. Another solution is based on the use of address registers to select two or more address rows. However, none of these mentioned architectures offers the two functionalities required to adopt the March IMC-8T test.

V. Conclusion

In this study, we first introduced March 8T-RPD, a novel test algorithm conceived for addressing isolated read port defects in 8T SRAM-based IMC architectures. With a complexity of 8N, this algorithm effectively meets these challenges. Subsequently, we expanded our work by integrating its capabilities with the March C- algorithm, resulting in March IMC-8T. Notably, March IMC-8T maintains an equivalent complexity to the existing March C-8 algorithm while achieving comprehensive coverage. It detects static faults, and isolated read port defects, even under worst-case conditions. However, to implement this dedicated algorithm, a DfT solution is crucial to embed new functions in the address decoder.

References

[1] S. Hamdioui et al., "Memristor based computation-in-memory architecture for data-intensive applications," Design, Automation & Test in Europe, 2015, pp. 1718-1725.

[2] A. Agrawal et al., "X-SRAM: Enabling In-Memory Boolean Computations in CMOS Static Random Access Memories," in IEEE Transactions on Circuits and Systems I: Regular Papers, vol. 65, no. 12, pp. 4219-4232, Dec. 2018.

[3] A. Jaiswal et al., "8T SRAM Cell as a Multibit Dot-Product Engine for Beyond Von Neumann Computing," in IEEE Trans. on VLSI Systems, vol. 27, no. 11, pp. 2556-2567, Nov. 2019.

[4] G. Singh et al., "A Review of Near-Memory Computing Architectures: Opportunities and Challenges," Euromicro Conference on Digital System Design, Prague, Czech Republic, pp. 608-617, 2018.

[5] A. Jaiswal et al., "8T SRAM cell as a multibit dot-product engine for beyond von Neumann computing," IEEE Trans. VLSI Syst., vol. 27, no. 11, pp. 2556-2567, Nov 2019.

[6] A. Biswas et al., "CONV-SRAM: An energy-efficient SRAM with in-memory dot-product computation for low-power convolutional neural networks," IEEE J. Solid-State Circuits, vol. 54, no. 1, pp. 217-230, Jan 2018.

[7] J. Zhang et al., "In-memory computation of a machine-learning classifier in a standard 6T SRAM array," IEEE Jour. of Solid-State Circuits, vol. 52, no. 4, pp. 915–924, Apr. 2017.

[8] https://www.eejournal.com/article/in-memory-computing/, In-Memory Computing, No Fewer than Four Approaches, by Bryon Moyer, 2019.

[9] M. Kooli et al., "Smart instruction codes for in-memory computing architectures compatible with standard SRAM interfaces," Design, Automation & Test in Europe Conference & Exhibition, 2018.

[10] M. Fieback et al., "Structured Test Development Approach for Computation-in-Memory Architectures," IEEE International Test Conference in Asia, Taipei, Taiwan, pp. 61-66, 2022.

[11] M. Fieback et al., "Testing Scouting Logic-Based Computation-in-Memory Architectures," IEEE European Test Symposium, Tallinn, Estonia, pp. 1-6, 2020.

[12] T.L. Tsai et al., "Testing of In-Memory Computing 8T SRAMs," Proc. IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems, 2019.

[13] L. Ammoura et al., "Preliminary Defect Analysis of 8T SRAM Cells for In-Memory Computing Architectures," 16th International Conference on Design & Technology of Integrated Systems in Nanoscale Era, pp. 1-4, 2021.

[14] L. Ammoura et al., "Intra-cell Resistive-Open Defect Analysis on a Foundry 8T SRAM-based IMC Architecture," IEEE European Test Symposium, Venezia, Italy, 2023, pp. 1-4.

[15] L. Ammoura et al., "Analysis of resistive defects on a foundry 8T SRAM-based IMC architecture," Microelectronics Reliability," Volume 147, 2023.

[16] S. Hamdioui et al., "Testing Computation-in-Memory Architectures Based on Emerging Memories," IEEE International Test Conference, Washington, 2019, pp. 1-10.

[17] J.-F. Li et al., "Testing of Configurable 8T SRAMs for In-Memory Computing," IEEE Asian Test Symposium, 2020.

[18] A. Bosio, et al., "Advanced Test Methods for SRAMs," ISBN 978-1-4419-0938-1, Springer, 2009.

[19] A.J. van de Goor et al., "Functional Memory Faults: A Formal Notation and a Taxonomy", VLSI Test Symposium, pp. 281-289, 2000.

[20] K. Monga et al., "A Novel Decoder Design for Logic Computation in SRAM: CiM-SRAM," IEEE India Council International Conference, Guwahati, India, 2021