2024 29th IEEE European Test Symposium (ETS)
- Industry Paper -

Combining Built-In Redundancy Analysis with ECC for Memory Testing

Luc Romain
Siemens Digital Industries Software
Ottawa, Canada
luc.romain@siemens.com

Paul-Patrick Nordmann
Siemens Digital Industries Software
Hamburg, Germany
paul-patrick.nordmann@siemens.com

Benoit Nadeau-Dostie
Siemens Digital Industries Software
Ottawa, Canada
benoit.nadeau-dostie@siemens.com

Lori Schramm
Siemens Digital Industries Software
Atlanta, GA, USA
lori.schramm@siemens.com

Martin Keim
Siemens Digital Industries Software
Orlando, FL, USA
martin.keim@siemens.com

2024 IEEE European Test Symposium (ETS) | 979-8-3503-4932-0/24/$31.00 ©2024 IEEE | DOI: 10.1109/ETS61313.2024.10567190

***Abstract*—Error Correction Codes (ECC) in current designs typically serve two purposes. For emerging non-volatile memory types (NVM), such as embedded magnetoresistive random access memory (eMRAM), ECC is necessary to counter the probabilistic behavior of the NVM, rendering the combined NVM/ECC a deterministic memory again. The second and much more prominent usage of ECC today is to protect the system against transient faults in the memory, here typically for SRAMs. On the other hand, new defect types for such emerging memories and new technology nodes may exceed reasonable costs of conventional row and column repair. To improve yield, users explore ECC as an option to augment the repair capability of the memory. This paper brings such an augmentation into a standard memory test and repair flow. It allows a user-defined, post silicon trade-off of using all or parts of the corrective power of the ECC for yield improvements and/or for system protection. Experimental results underline the very low area cost of this augmentation.**

***Index Terms*—Error Correction Codes (ECC), Built-In Redundancy Analysis (BIRA), Built-In Self Repair (BISR)**

I. Introduction

Since NVM technology is still under development, ECC logic and redundant elements are commonly used to improve device reliability [1], [2]. The test and repair solution should offer the flexibility to adjust the number of defects to be corrected using ECC and to collaborate with the Built-In Redundancy Analysis (BIRA) circuitry during memory manufacturing test. Using ECC correction and redundant elements during manufacturing is proposed in [3]. However, most of the techniques rely on failure data collection and off-chip post-processing to determine the final redundant element allocations. This paper describes an autonomous on-chip solution consisting of an augmented BIRA circuitry capable of calculating an optimal repair solution using both redundant elements and ECC, given user-defined requirements. An overview of the standard repair test flow is shown in Section II. The augmented BIRA circuit, referred to as ECC-aware BIRA, is presented in Section III. The built-in solution uses the memory's ECC correction capability and redundant elements to improve manufacturing yield, especially for NVMs. It can also be used with ECC-equipped SRAMs during in-system memory testing. Furthermore, we show a test sequence with fault injection where the ECC-aware BIRA hardware automatically allocates redundant elements in an optimal way. Simulation results are provided to demonstrate the functionality of the proposed solution. Finally, we analyze the area and performance of the solution using a logic synthesis tool in SectionIV.

II. Standard Memory Repair Flow

The memory test hardware involved in testing memories with redundant elements is shown in Figure 1. The memory interface is connected to the memory, on the right, and the memory test controller is on the left. A BIRA module is located inside the memory interface. The elements in red illustrate the new ECC-aware components, which will be introduced in Section III-B. These elements do not participate in the standard memory test flow. The comparator registers

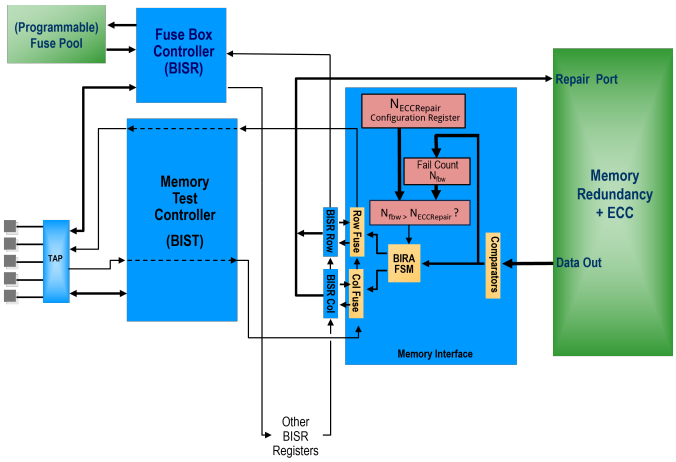

Figure 1: Built-in memory test architecture overview

capture the defect locations during the memory test execution. The BIRA logic uses the values of these registers to calculate a

repair solution which is stored into the fuse registers. Each fuse register is associated to a redundant element in the memory.

A manufacturing test and repair flow is shown in Figure 2 and described in [4]. The goal of the repair flow is to categorize devices as ”good” or ”bad”. Devices are classified as ”bad” when they have defects that cannot be repaired or that prevent the test flow from executing normally. Devices that are classified as ”good” either have no defects or can be repaired. The blue boxes contain test patterns related to the Built-In Self-Repair (BISR) controller which is responsible for exchanging repair information with the BIRA modules and performing fuse box programming. The green boxes contain test patterns that perform the memory test and BIRA execution. The test patterns are labeled TP0 to TP3.*n* where the *n* is a group index. Designs with a very large number of memory test controllers often split the TP3 patterns into *n* smaller groups of controllers. The generation of the TP* test patterns is such that a binary ”pass” or ”fail” status can be inferred based on their execution result. A test pattern is said to ”pass” when there are no compare failures, or ”fail” when one or more compare failure(s) are reported. When a patterns fails, the interpretation of the compare mismatches is not required by the flow.

Test pattern TP0 (BISR Power-Up) is executed first and the subsequent patterns are executed according to the flow in Figure 2 based on the pass/fail status of each pattern.

Test patterns TP1.*n* apply the memory test algorithms. The BIRA is enabled and calculates a repair solution when errors are detected. Repair allocation and status bits are available on output ports of the memory interface at the end of the execution. A ”fail” status from the TP1.*n* indicates the memory is not repairable; a ”pass” status indicates the memory is either repairable or fault-free.

Test pattern TP2.1 determines if one or more BIRA module allocated redundant elements during test pattern TP1.*n*. The pattern passes if no repair is needed or fails if one or more memory needs to be repaired. If repair is needed, the repair solution is transferred from the BIRA to the BISR registers in TP2.2a and programmed into the fuse box in TP2.2b. Test pattern TP2.3 verifies that the repair solution is correctly programmed in the fuse box. The verification is performed by comparing the content of the BISR registers against the content that is programmed in the fuse box. The repair solution is applied on the memory repair ports after running test pattern TP2.3. Test patterns TP3.*n* perform a final verification of the memories using the redundant elements. Any defect within the redundant elements would be detected at this time.

III. ECC-AWARE MEMORY REPAIR FLOW

Adding ECC logic to memories is done during the design phase. This can be done using different methods. One method is to generate and insert the ECC logic within the functional design near the memory. In this case the memory test logic can directly intercept the data input and output ports on the memory, as shown in Scenario 1 of Figure 3. Another method is to use a memory compiler capable of generating ECC logic within the memory module, as shown in Scenario 2. An ECC

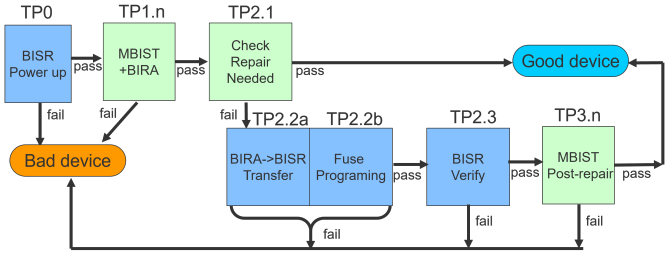

Figure 2: Standard Memory BIST test and repair manufacturing flow

enable port must be present on the memory module to allow deactivation of the embedded ECC logic.

N_{ECC} denotes the number of correctable bits per word implemented by the ECC logic. This parameter is provided in the memory datasheet.

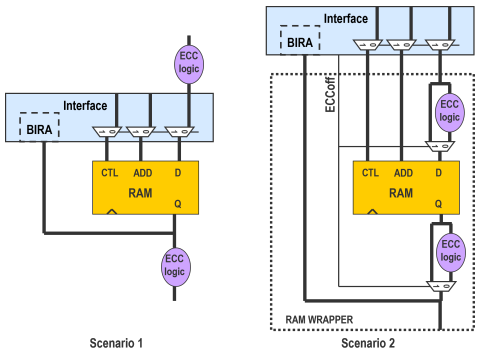

Figure 3: ECC implementation methods

The $N_{ECCRepair}$ is a user-configurable setting described in Section III-B, which is initialized inside the memory test circuit before running the memory test patterns TP1.*n* and TP3.*n*. It specifies a threshold value for the maximum number of defective bits within a word to be delegated for correction by the ECC and thus ignored by the BIRA logic. Devices that use a non-zero threshold value during memory testing will rely on ECC to correct hard defects when operating in system mode. For NVMs, $N_{ECCRepair}$ may be set to $N_{ECC} - 1$ to improve yield during manufacturing test. For example, setting $N_{ECCRepair}$ to 1 for a memory with $N_{ECC} = 2$ allows words with single-bit defects to be corrected by ECC while preserving one bit of ECC correction for soft errors. Redundant elements are allocated when encountering words with two defects or more. For SRAMs, the $N_{ECCRepair}$ is generally set to 0 during manufacturing and to N_{ECC} during in-system functional memory test. Activating the ECC-aware BIRA mode during in-system testing enables the memory BIST controller to be tolerant of defects that are correctable by ECC. Output status ports on the ECC-aware BIRA circuit indicate if defects are detected and if they are correctable by ECC. This information can be used in the system to determine the appropriate actions.

When testing memories with ECC enabled, two potential types of test escapes must be considered. These test escapes are categorized as Type-1 and Type-2 [4] and are described in the next section.

A. ECC Test Escape Handling

A Type-1 test escape can occur when a stuck-at-one (SA1) and a stuck-at-zero (SA0) defect are simultaneously present within a memory word. Applying the all-zero or all-one pattern will only reveal either the SA1 or the SA0 defect, respectively, but not both simultaneously. Applying checkerboard or inverse-checkerboard patterns may also fail to detect both defects simultaneously if the two defects are on odd/even bit positions. Without the appropriate test logic, a BIRA circuit could interpret these as single-bit defects and incorrectly assume that the ECC logic will handle them.

In order to address these Type-1 test escapes, multiple data patterns must be applied at the reference address, i.e. the address under test. For each reference address, at least two data patterns must be applied to test against SA0 and SA1 defects. The memory test controller is enhanced to accumulate the test data results for each data pattern applied at the reference address. This allows the ECC-aware BIRA logic to compound all defects found at the reference address and make the appropriate decision whether to allocate a redundant element or to use ECC.

Type-2 test escapes can occur under certain conditions when the ECC logic is enabled during test. When ECC logic is enabled, additional code bits are computed and stored along with the data bits inside the memory. Conventional test patterns applied to the memory such as all-zero, all-one, checkerboard and inverse_checkerboard causes some ECC code bits to never toggle. Therefore, if a code bit is affected by a defect, the memory test would fail to detect the problem, but the defect would likely manifest itself during functional operation when arbitrary data is written to the memory. Solutions to address Type-2 test escapes are described in [4].

When the ECC logic is integrated as shown in Scenario 2 of Figure 3, the memory data is processed by the ECC encoding and decoding logic. If ECC is enabled during the memory test, incorrect data bit values caused by defects inside the memory array are corrected by ECC and are not visible to the memory test controller. Also, if a memory word has more defects than what can be corrected, the ECC logic may generate inconsistent data on the memory output. Furthermore, the failure data processed by the BIRA logic will likely not correspond to the actual defect locations.

B. ECC-aware Design and Operation Details

In this section we describe the differences between a standard BIRA circuit used with memories having redundant elements only, and the ECC-aware BIRA circuit used with memories having redundant elements and ECC. The logic implementing the ECC-aware functionality is illustrated in Figure 1. The BIRA circuit is activated in test pattern TP1.*n* shown in Figure 2. The the repair solution is calculated in this step.

When a memory has ECC and redundant elements, like the one shown in Figure 1, the memory interface is augmented with three components shown in red: a configuration register, a failure counter, and a threshold comparator. These modules combined with the BIRA logic become the ECC-aware BIRA logic.

The run-time configuration register stores the $N_{ECCRepair}$ threshold value to use during the memory test execution. The allowed $N_{ECCRepair}$ value ranges from 0 to the maximum number of ECC correctable bits, N_{ECC}. A failure counter module calculates the number of comparator registers that captured defects during the memory read operations. This number corresponds to the number of failing bits detected inside the word (N_{fbw}) at the current reference address. Setting $N_{ECCRepair}$ to 0 deactivates the ECC-aware functionality and instructs the BIRA logic to exclusively use redundant elements to repair all defects. Setting $N_{ECCRepair}$ to a value of 1 or more activates the ECC-aware functionality.

The repair sequence shown in Figure 4 illustrates the steps that are implemented inside the memory test logic to efficiently perform ECC-aware BIRA analysis with a single BIST controller execution.

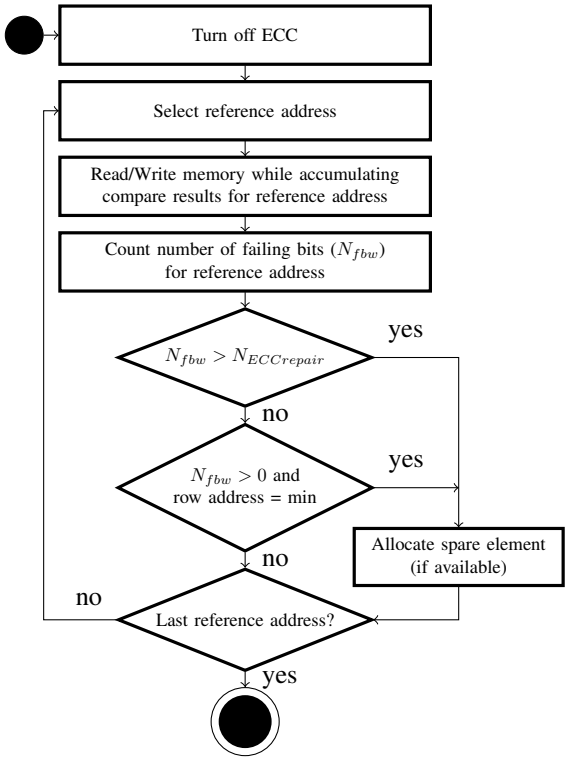

Figure 4: ECC-aware BIRA logic sequence diagram

The ECC logic is deactivated to allow the memory test logic to observe all memory defects. An algorithm specifically designed to avoid Type-1 and Type-2 ECC test escapes is

selected. Examples of such algorithms are shown in Section III-C.

These algorithm iterate over all memory addresses which are designated as the reference address. For each iteration, test data patterns are applied, compared, and accumulated. If $N_{fbw} > N_{ECCRepair}$, the number of failing bits exceeds the specified threshold and a redundant element is allocated by the BIRA logic, if available. When defects occur on the first row address, a redundant element is systematically allocated even when $N_{fbw} \leq N_{ECCRepair}$. Since bit-line defects account for 42.3% of the memory defects [3], the strategy of allocating a redundant column element in this case could be beneficial. Because bit-line defects propagate to all rows, a defect in the first row address has a high probability of being a bit-line defect. A redundant element is systematically allocated in this case. This strategy avoids relying on the ECC logic to systematically correct the failure in all rows. Using a redundant element is more efficient and preserves the ability for the ECC logic to correct future defects.

Multi-bit failures ($N_{fbw} > 1$) are typically handled by allocating a redundant row. If the memory has a redundant IO block, it will be allocated for consecutive multi-bit failures instead of a redundant row. If no redundant element is available or the remaining ones cannot be used for the specific defects, the memory is declared non-repairable.

Although it would be technically feasible to allocate multiple redundant columns to repair multi-bit failures, the BIRA logic required to calculate the repair solution would be area intensive. For this reason, the BIRA logic is unable to allocate multiple redundant column elements when encountering a multi-bit error. If a redundant row element is available, it will be allocated; otherwise, the memory is declared non-repairable.

C. ECC-aware BIRA algorithms requirements

All combinations of stuck-at defects within a word are detected using the well-known PMOVI algorithm [5]. This algorithm is implemented using the following sequence:

```
1 // All cells initialized with 'd'
2 for add in Addresses() {
3   add.Read(d);
4   add.Write(d_inv)
5   add.Read(d_inv)
6 }
```

Listing 1: PMOVI Algorithm sequence

`d` is an arbitrary data pattern and `d_inv` is its inverse. The `Write(<argument>)` function performs a memory write operation with the specified value. The `Read(<argument>)` function performs two actions: it executes a read operation on the memory and then compares the data against the specified `<argument>` value. The result is accumulated inside the comparator registers.

This sequence shown in Listing 1 is repeated four times where all combinations of data, inverted data, incrementing and decrementing addresses are performed. However, this algorithm does not cover all cases where errors at the reference address are induced by operations on surrounding cells. Other common algorithms, such as BitSurroundDisturb, GalColumn, and GalRow [6], [7] are good alternatives to improve the defect coverage of the reference cell.

The BitSurroundDisturb algorithm implements the following sequence:

```
1  for d in {0 1} {
2    for add in Addresses() {
3      // Fill background with 'd'
4      add.Write(d);
5    }
6    for refAdd in Addresses() {
7      for surAdd in refAdd.Surround() {
8        surAdd.Read(d);surAdd.Write(d_inv)
9        refAdd.Read(d)
10     }
11     // Restore surrounding cells with 'd'
12     for surAdd in refAdd.Surround() {
13       surAdd.Read(d_inv);surAdd.Write(d)
14       refAdd.Read(d)
15     }
16   }
17 }
```

Listing 2: BitSurroundDisturb Algorithm

The `Surround()` function returns a list of addresses that physically surround the reference address. When using ECC-aware BIRA, the algorithm must be modified to leverage the memory test controller's ability to accumulate test results for the reference address only. This modification is necessary to cover the Type-1 test escapes during the pre-repair testing phase. All data compares on the surrounding addresses must be removed, and only the data of the reference address is compared. Applying the read and write operations on the surrounding addresses creates activity and increases the test coverage of the reference address.

The `surAdd.Read(<arg>)` operations on lines 8 and 13 of Listing 2 must be replaced by `surAdd.Read(X)`. The special value `X` prohibits the compare step allowing the content of the comparators to remain unchanged. `Read(X)` operations are performed on address locations surrounding the reference address. This creates activity around the reference cell without changing the content of the comparator registers. Only the defects found on the reference address get accumulated inside the comparator registers.

IV. Simulation Results

A simulation demonstrating the ECC-aware BIRA functionality is performed using a design that contains one 1RW_128x8 SRAM memory instance with two ECC correctable bits. This memory has one redundant column element and two redundant row elements. The $N_{ECCRepair}$ threshold value is configured to 1. This instructs the ECC-aware BIRA

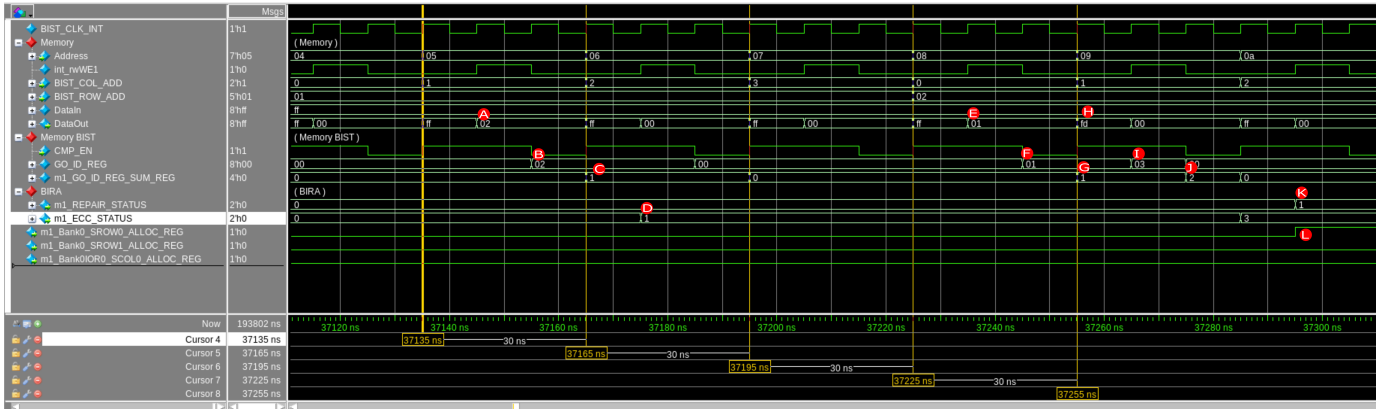

Figure 5: Waveforms of ECC-aware repair simulation with fault injection

logic to use ECC for words having zero or one defects; words with two or more defects will trigger the allocation of a redundant element by the BIRA logic. The defects listed in Table I are injected simultaneously inside a memory at the start of the simulation. The simulation waveform results are shown

Table I: Fault injection location and type

Index	Address[6:0]	IO	Fault
F1	7'h5	Q[1]	Stuck at 1
F2	7'h8	Q[0]	Stuck at 1
F3	7'h8	Q[1]	Stuck at 0

in Figure 5. Specific events in the waveforms are annotated using circled letters (A) to (L). The respective details of these events are described the following paragraphs.

The fault F1 is observed at address 7'h5 at the event shown by marker (A) in Figure 5, and it is captured by the BIST controller at (B) inside the error accumulator registers (GO_ID_REG). Since this is a single-bit failure, the summing circuit contains the value 1 at (C). Because the number of defects (N_{fbw} = m1_GO_ID_REG_SUM_REG = 1) does not exceed the specified $N_{ECCRepair}$ = 1 value, the ECC_STATUS register is set to 1'b1 at (D), indicating that ECC correction is needed.

The fault F2 is observed at address 7'h8 at (E) and captured by the BIST controller at (F) inside the comparator error accumulator registers. The summing circuit contains the value 1 at (G). A subsequent fault, F3, occurs at (H) at the same memory address but on a different memory IO. This sets a second error accumulator register to a logic high value at (I). At this time, the summing circuit increases to 2 at (J). This value exceeds the specified $N_{ECCRepair}$ value; therefore, the ECC logic cannot correct all defects in the current word. In addition, a redundant column cannot be allocated since this is a multi-bit failure. Therefore, a redundant row is allocated at (L), and the memory repair status (m1_REPAIR_STATUS) is set to 2'b01 (Repairable) at (K).

The processing of the memory failures by the ECC-aware BIRA circuit is pipelined to optimize performance. This explains why the error detection and ECC-aware redundancy analysis extends several clock cycles after the memory read operation.

V. Area and Performance

The analysis in this section focuses primarily on the relative differences between the circuit with ECC-aware repair compared to the baseline circuit. The baseline circuit used for this comparative analysis contains the memory interface with the BIRA logic without the additional ECC-aware logic.

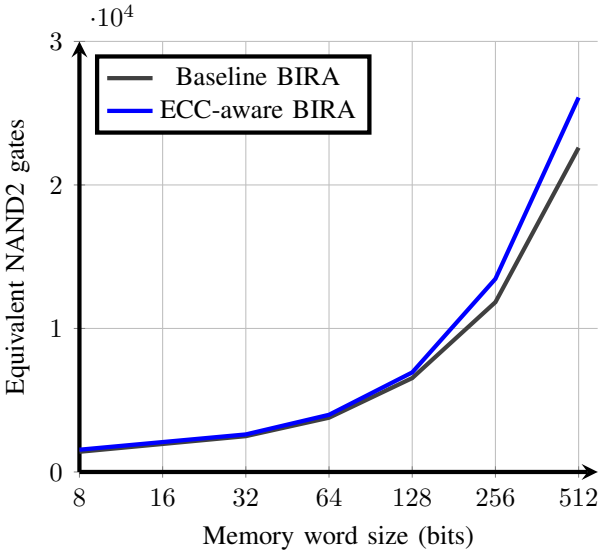

Figure 6: Area trend

Synthesis is performed on the memory interface modules using a 7nm library. The memory interface modules contain the memory test and BIRA logic. For simplicity, a statistical wire load model is selected for the analysis. The test design is composed of six memory instances with 8, 32, 64, 128, 256 and 512-bit word widths, respectively. Each memory has two redundant rows and one redundant column. The baseline circuit is then modified to include the ECC-aware BIRA logic

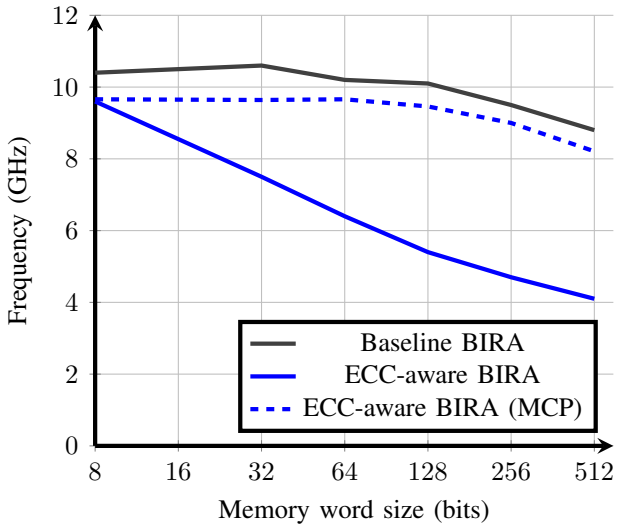

Figure 7: Maximum BIST clock frequency

and is compared against the baseline circuit. The number of ECC correctable bits is two for all memories.

Figure 6 shows the NAND2 equivalent synthesized area of the memory interfaces with BIRA only (baseline) and with the ECC-aware BIRA. The addition of the ECC-aware functionality contributes to an area increase of 10% for the 8-bit memory and of 15% for the 512-bit memory. When including the area of the memory test controller logic, the effective area increase ranges from 4.5% to 9.5% for 8-bit to 512-bit wide memories, respectively.

Figure 7 shows the maximum clock frequency achievable during synthesis for the same memory interface circuits. The frequency for the baseline and ECC-aware BIRA circuit are shown by the solid black and blue lines, respectively. The critical timing path reported during synthesis is from the error accumulator to the error summing circuit. The depth and complexity of the error summing circuit increases logarithmically with the memory word size. Therefore, the maximum clock frequency achievable during synthesis is impacted due to this additional circuitry. The maximum clock frequency of the ECC-aware BIRA circuit is reduced by 7.7% and 47% for 8-bit and 256-bit memories, respectively. Only the frequency of the BIST circuit is affected; the functional frequency remains unchanged.

The overall performance, however, can be improved with multi-cycle paths (MCPs) when using many common memory test algorithms. For example, algorithms that perform read operations of type read-modify-write do not apply data compares on consecutive clock cycles. When using such algorithms, critical paths in the ECC-aware circuit can be relaxed as a two-cycle MCP. This improves the maximum clock frequency of the ECC-aware circuit as shown by the blue dashed line. Here, the maximum clock frequency increases to 93% of the baseline circuit.

If memory algorithms applying data compares on consecutive clock cycles are used, the MCP can still be applied during synthesis to improve the circuit's performance. However, the memory test controller circuit must be configured to mask the odd or even data compares. Specifically, a first test is performed where the even data compares are skipped; then, a second test is done where the odd data compares are skipped. The ECC-aware BIRA circuit accumulates the results from both tests. This strategy doubles the test time, but it provides two clock cycles for the summing circuit to process errors.

VI. Conclusion

A test and repair flow for memories with redundant elements and ECC is presented. This flow is implemented using an ECC-aware BIRA circuit that can be configured to allocate redundant elements or use ECC correction to improve yield. This solution can be used to improve manufacturing yield, especially for NVMs, and it can also be used to test ECC-equipped SRAMs during in-system tests.

Simulation results with fault injection demonstrates how redundant elements are automatically allocated by the ECC-aware BIRA logic for the defects that are not corrected by ECC.

Synthesis shows an area increase ranging from 4.5% (8-bit memory) to 9.5% (512-bit memory) when adding the ECC-aware functionality into the BIRA logic compared to the overall test circuit without ECC-aware logic. Synthesis results also show that the maximum frequency of the memory test logic can operate up to 93% of the baseline clock frequency when adding the ECC-aware logic.

References

[1] Younggeun Ji et al., "Reliability of 8Mbit Embedded-STT-MRAM in 28nm FDSOI Technology", IEEE International Reliability Physics Symposium, 2019

[2] Younggeun Ji et al., "Reliability of Industrial grade Embedded-STTMRAM", IEEE International Reliability Physics Symposium, 2020

[3] Tze-Hsin Wu et al., "A Memory Yield Improvement Scheme Combining Built-In Self-Repair and Error Correction Codes", Proc. IEEE International Test Conference, 2012

[4] Cyrille Dray et al., "Transitioning eMRAM from Pilot Project to Volume Production", IEEE International Test Conference, 2023

[5] A.J. van de Goor, "The Effectiveness of Scan Test and its New Variants", International Workshop on Memory Technology, Design and Testing, 2004

[6] Said Hamdioui et al.,"March SS: A Test for All Static Simple RAM Faults", IEEE International Workshop on Memory Technology, Design and Testing, 2002

[7] ,A.J. van de Goor,"Disturb Neighborhood Pattern Sensitive Fault", IEEE VLSI Test Symposium, 1997