International Journal of Electrical and Computer Engineering (IJECE)
Vol. 14, No. 1, February 2024, pp. 140~156
ISSN: 2088-8708, DOI: 10.11591/ijece.v14i1.pp140-156

Memory built-in self-repair and correction for improving yield: a review

Vijay Sontakke, Delsikreo Atchina
Networking and Edge Group, Intel Corporation, Allentown, United States of America

Article Info

Article history:

Received Mar 30, 2023
Revised Jul 20, 2023
Accepted Aug 1, 2023

Keywords:

Built-in repair analysis
Built-in self-repair
Built-in self-test
Error correcting codes
Memory testing
Yield

ABSTRACT

Nanometer memories are highly prone to defects due to dense structure, necessitating memory built-in self-repair as a must-have feature to improve yield. Today's system-on-chips contain memories occupying an area as high as 90% of the chip area. Shrinking technology uses stricter design rules for memories, making them more prone to manufacturing defects. Further, using 3D-stacked memories makes the system vulnerable to newer defects such as those coming from through-silicon-vias (TSV) and micro bumps. The increased memory size is also resulting in an increase in soft errors during system operation. Multiple memory repair techniques based on redundancy and correction codes have been presented to recover from such defects and prevent system failures. This paper reviews recently published memory repair methodologies, including various built-in self-repair (BISR) architectures, repair analysis algorithms, in-system repair, and soft repair handling using error correcting codes (ECC). It provides a classification of these techniques based on method and usage. Finally, it reviews evaluation methods used to determine the effectiveness of the repair algorithms. The paper aims to present a survey of these methodologies and prepare a platform for developing repair methods for upcoming-generation memories.

Corresponding Author:

Vijay Sontakke
Networking and Edge Group, Intel Corporation
1110 American Pkwy, Allentown, PA, 18109, USA
Email: vijay.sontakke@intel.com

1. INTRODUCTION

Due to dense structures, memories are highly prone to manufacturing defects and failures during system operation. They experience permanent hard errors that may be caused by the manufacturing process, physical stress resulting from environmental conditions, or aging [1]. In addition, soft errors show up in memories during the run time due to the interaction of high-energy particles with the silicon substrate [2]. Memory failures are top fail bins in typical production testing, and they need to be addressed promptly to improve the yield of system on-chip (SOC). Redundancies are incorporated in memories to help to improve yield. Additionally, error correcting codes (ECCs) are used to address soft errors seen in them during system operation.

ISO standard 26262 defines guidelines for the functional safety of automotive systems. One crucial requirement is detecting latent faults seen in devices during automotive operation. The requirement mandates performing testing periodically during functional operation. Memories are used in enterprise servers, and fault-free memories are required for their reliable operation. Hence, they, too, demand periodic checking.

Built-in repair analysis (BIRA) is commonly used to repair memories post-manufacturing. First, redundancies in the form of spare elements are added to memory cores. Then, the BIRA searches for the replacement of the fault elements with healthy spare elements. During system operation, addresses are

translated to choose spare elements in place of faulty elements. Information on faults is collected by running memory tests. Redundancy analysis (RA) determines the allocation of spares to repair these cells.

Two methods exist for defect localization of faulty memory chips-software-based and hardware-based. The software-based method uses a bitmapping procedure that scans out fault information using a built-in self-test (BIST) controller. This information is further analyzed to know the physical location of the faulty cell. Hardware-based techniques use electrical failure analysis. They typically use photon emission [3] and laser [4]. Software-based diagnosis is preferred as mainstream due to its faster turnaround time and accuracy. It is a well-established technique to detect faults affecting single or multiple bits in memory. However, the accuracy of software-based techniques drops for peripheral faults or soft errors. For such cases, hardware-based techniques could be used as a backup. They are also helpful when bitmapping or software-based techniques cannot be set up due to a lack of resources like scrambling information.

Memories show different failures at different instances in time. For example, while infant failures are seen during early usage after manufacturing, wear-out, and aging errors show during later usage [5]. The probability of such errors and techniques used to avoid failures resulting from them are shown in Figure 1.

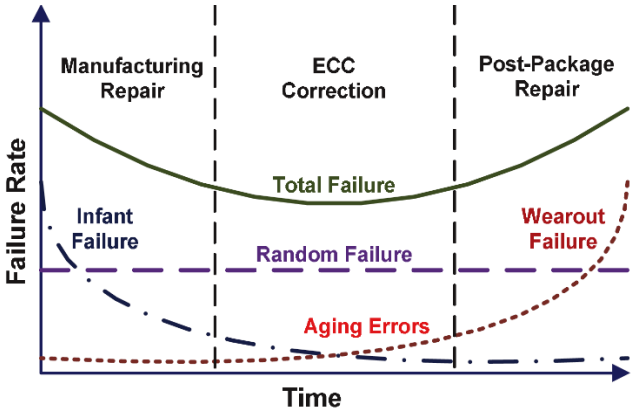


Figure 1. Bathtub curve showing aging errors in dynamic random access memory (DRAM) [5]

Researchers have proposed multiple methodologies to repair memories and use correcting codes and simulators to evaluate their performance. This paper describes the principles, algorithms, and architecture details used in them. It classifies the techniques, discusses the advantages and disadvantages of each type of those techniques, and enables selection for particular usage.

The organization of the remaining sections of this paper is as follows. Section 2 gives a background of memory repair. Then section 3 classifies the repair techniques based on the principle used and its application. Details of the repair methodologies and their applications are discussed in section 4. Section 5 presents types of repair analysis, while section 6 provides details of simulators used for repair efficiency evaluation. Finally, section 7 discusses the pros and cons of the techniques, and section 8 concludes the paper. Few abbreviations and acronyms are used throughout this paper. Table 1 gives their full form.

Table 1. Abbreviations and acronyms

Abbreviation	Full form	Abbreviation	Full form
3D	Three dimensional	ATE	Automatic test equipment
TSV	Through-silicon-via	HBM	High bandwidth memory
ECC	Error correcting code	CAM	Content addressable memory
BIRA	Built-in repair analysis	LUT	Look-up-table
BISR	Built-in self-repair	BOSR	Built-off self-repair
BIST	Built-in self-test	PPR	Post package repair
SEC-DED	Single error correction, double error detection	LADA	Laser assisted device alteration
RA	Repair algorithm	PEM	Photon emission microscopy
OTP	One-time programmable		

2. BACKGROUND

The performance of the BIRA methodology is indicated by the repair rate [6]. The repair rate indicates the probability of finding a repair solution for a given set of faults. It is defined as (1).

$$Repair\ Rate = \frac{No.of\ repaired\ chips}{No.of\ total\ tested\ chips} \quad (1)$$

The repair rate also includes memories that cannot be repaired with any solution using available spares. Therefore, it does not help to compare the efficiencies of different RA methodologies. Instead, another rate called normalized repair rate is used for this purpose which considers only memories which can be repairable.

$$Normalized\ Repair\ Rate = (No.of\ repaired\ chips)/(No.of\ repairable\ chips) \quad (2)$$

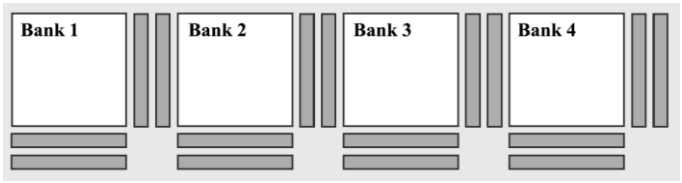
Repair analysis is performed using automatic test equipment (ATE) or built-in self-repair (BISR) logic present on the chip. Most of the commercial memories are tested directly by ATE. When ATE performs memory testing, it transmits test patterns to the chip and receives response data. For the failed location, fault information is stored in ATE memory. Once the test algorithm finishes, ATE analyzes the faults and finds repair solutions. Finally, the solution is sent back to the chip, which performs the replacement of the fault cell with spares.

As embedded memories are difficult to access directly by ATE [7], they are tested and repaired by adding BIST and BIRA controllers, respectively. BIST engines execute algorithms to check faults in memories. The fault information is stored on-chip and analyzed by BIRA logic to find a repair solution. Later, the repair solution is used for the replacement of faulty elements. Memories are tested and repaired without depending on ATE, and so it is called self-repair.

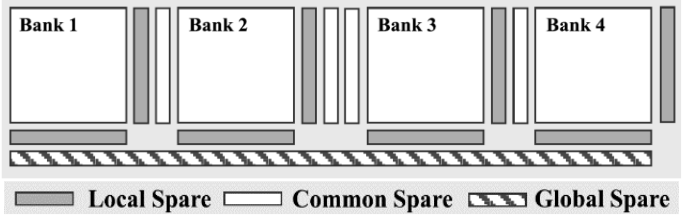
2.1. Type of redundancy spares

Many conventional memories employ single redundant rows or columns to repair cells that are found defective. Finding a spare row/column allocation solution is an nondeterministic polynomial (NP) complete problem [8]. With memory arrays' increasing size and density, using the simple redundancy structure is inefficient. The time required to complete redundancy analysis increases dramatically and limits the repair rate with them. This section summarizes the types of spares proposed by various researchers.

A redundancy scheme adds spare elements to memory banks, as shown in Figure 2. Figure 2(a) shows a memory employing simple row and column spare as a redundant element. Kim *et al.* [9] presented a redundancy structure with additional common columns and global rows, as shown in Figure 2(b). While the local spare could be used to repair a fault in the same memory bank, a common spare can be used for faults in adjacent banks. In addition, the global element could be used to repair faults in multiple memory banks with the same row address.



(a)



(b)

Figure 2. Redundancy schemes with multiple type of spare elements [9] (a) using simple row and column elements as spares and (b) using variety of spare elements

In study [10], common and global spares for both rows and columns are added, as shown in Figure 3. The author proposes to allocate spares starting with local, then common, and finally, the global type for the remaining faults' repairing. A high repair rate could be achieved using such an allocation order. Architectures

presented in [11] further increase the count of each local, common, and global spare to have more flexibility in allocation. Techniques were also proposed which use static random-access memory (SRAM), dynamic random-access memory (DRAM), or content-addressable memory (CAM) as spares.

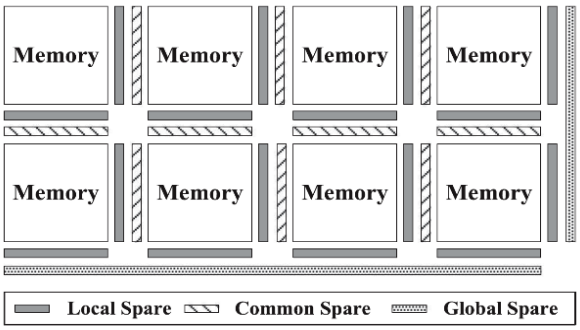


Figure 3. Redundancy scheme with common and global spares for both rows and columns [10]

2.2. Previous surveys on memory repair

A survey on memory built-in self-repair architectures is presented in [12]. It mainly surveys memory testing methods and contains a small section on repair architectures. It discusses the essential operation of repair and sharing of BISR logic. Later, Cho *et al.* [13] presented a survey on repair algorithms. It discussed various spare architectures and repair algorithms, including comparing their characteristics. These surveys are presented in the year 2013 and 2016, respectively. Since then, multiple methodologies and algorithms have been presented by researchers. This paper presents a survey of those methodologies and algorithms.

3. CLASSIFICATION

Over the years, researchers have proposed various methodologies for repairing memories. In addition, many techniques for using correction codes are also presented. This survey covers the methodologies presented in recent times. Table 2 shows variants of methodologies based on the fundamental principle used and their application. Table 3 lists variants of repair analysis types, while Table 4 has a list of various simulators.

Table 2. Methodologies for repair and error corrections

Sr.	Methodology	References
1	3D memories repair	[14]–[17]
2	In-field repair	[17]–[20]
3	Error correction code (ECC) usage	[18], [20]–[25]
4	CAM usage for repair analysis	[10], [14], [26]–[31]
5	SRAM usage for repair analysis	[15], [32]–[36]
6	DRAM usage for repair analysis	[37]
7	Using separate memory for redundancy instead of spare rows/columns	[16], [17], [21], [24], [34], [38]–[46]
8	Standardization approach	[47]–[49]
9	Read only memory (ROM) error corrections	[22], [23]
10	Hardware methods for defect localization	[50], [51]

Table 3. Repair analysis types

Sr.	Technique	References
1	Dynamic repair analysis	[28], [52]
2	Off-chip repair analysis	[9], [15], [49], [52]
3	Early termination repair analysis	[9], [53]
4	Improving repair analysis clock frequency	[19], [54]

Table 4. Simulators

Sr.	Technique	References
1	Simulators for measuring reliability and repair efficiency	[5], [55]–[58]

4. METHODOLOGIES FOR REPAIR AND ERROR CORRECTIONS

4.1. Three-dimensional (3D) memories repair

High bandwidth memory (HBM) architectures use independent interfaces called channels. Memory dies are stacked and connected using the channels to perform a high-bandwidth operation. The channels can be accessed concurrently [15]. The memory dies are attached to a logic die, which performs test operations like repair. The logic die contains blocks like memory controller, ECC, memory BIST, and BISR logic. The individual memory dies are tested and repaired on ATE. These memory dies are connected with each other and with the logic die using through-silicon vias (TSVs) and micro-bumps. These new elements and connections may lead to new defects post-bonding. Also, memory arrays may show new types of defects. This section summarizes testing and repairing such defects.

Kang *et al.* [14] presented a repair scheme to improve yield for 3D memories. As shown in Figure 4, it contains a BISR module in the base die. Individual memory dies do not contain BISR; they are tested and repaired on ATE. In contrast, memories in the base die are tested and repaired using the BISR present in it. In the post-bond test procedure, the TSVs are used as test buses, and the memory dies are tested in parallel using them. The failing memory dies are repaired sequentially since only one BISR is present. Initially, the BIRA module collects information about available redundancy in each memory. After testing is completed, fault information is available in CAM, which is present in the BIRA module. RA algorithm performs an exhaustive analysis of the faults with all combinations of rows and columns. The dies are repaired in descending order of the number of faults found. The possibility of finding a repair solution decreases for memories with more faults. The memory becomes irreparable, and consequently, the complete package becomes faulty. So, identifying such memories save repair time as other dies need not be tested further. Also, since tests and repairs are done only for faulty dies, fault information storage requirements reduce, which decreases area overhead.

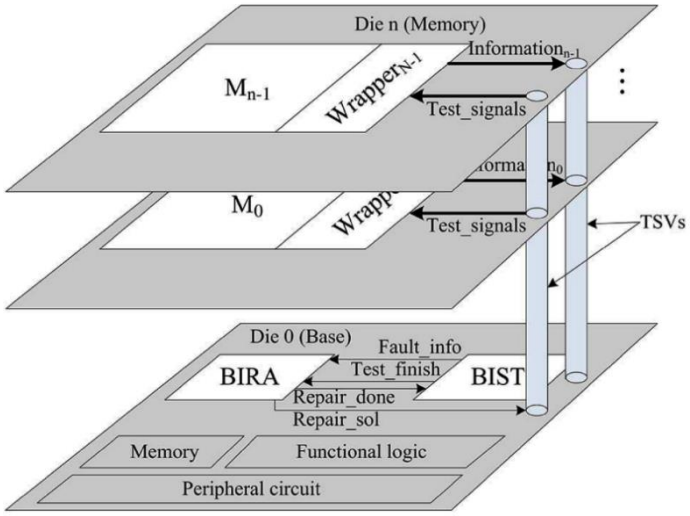


Figure 4. 3D BISR architecture [14]

Liu *et al.* [15] presented another architecture to test channel-based DRAM memories arranged in a 3D stack. The base die contains repair control logic consisting of SRAM for redundancy, allocator, and channel controllers. The channel controllers work concurrently. The allocator determines the channel to SRAM locations assignment for repairing faulty lines in DRAM memories. When BIST detects a new fault, the BISR controller requests the allocator to allocate a location in SRAM. The allocator returns the SRAM location's address, which the BIRA module uses to update the look-up-table (LUT).

Area overhead is low with this built-off self-repair (BOSR) scheme. For an 8 GB memory stack with 8×1 GB configuration, the total BOSR area is 0.72%, respective to DRAM size. Area reduces if fewer channels are used. The critical timing path is related to the address comparison logic in the BIRA module. The latency added by the critical path is negligible compared to DRAM read latency. The author further suggests using CAM instead of SRAM to improve this latency. The author compared BOSR with a conventional repair approach based on a redundant row/column in DRAM itself. It used various sizes of SRAM for redundancy (4, 8, and 16 M) with different DRAM sizes (1, 4, and 8 GB). BOSR outperforms for repair rate as DRAM size increases.

Lin *et al.* [16] presented a memory repair scheme for 3D stacked memories. DRAM dies contain redundant elements, as shown in Figure 5. Local redundancy is used to repair faults during ATE-based testing of the individual dies. The reconfiguration is programmed using fuses. When dies are stacked, the test mode of DRAM dies is not activated. So, BIST in the logic die tests and repairs any faults occurring post-stacking. Whenever a fault is detected, the BISR controller analyzes it and maps the fault line to global redundancy. Usage of global spare is done in sequential order. First, spares from die0 are used, followed by spares in die 1, die 2, and so on. This scheme has a performance penalty since addressing comparison results in a larger timing path, including DRAM access time and delays in compare logic. The author presented another architecture to improve the performance by implementing global redundancy by adding an SRAM in the logic die. The access time of SRAM is shorter, which results in a shorter timing path. Any faulty cells in DRAM are remapped to the SRAM locations.

Another architecture to improve the yield of 3D stacked memories is presented in [17]. It uses spare memory available in DRAM itself to replace any faulty bits. It also suggests implementing redundancy using extra rows/columns for future DRAM as per post package repair (PPR). in the joint electron device engineering council (JEDEC) standard.

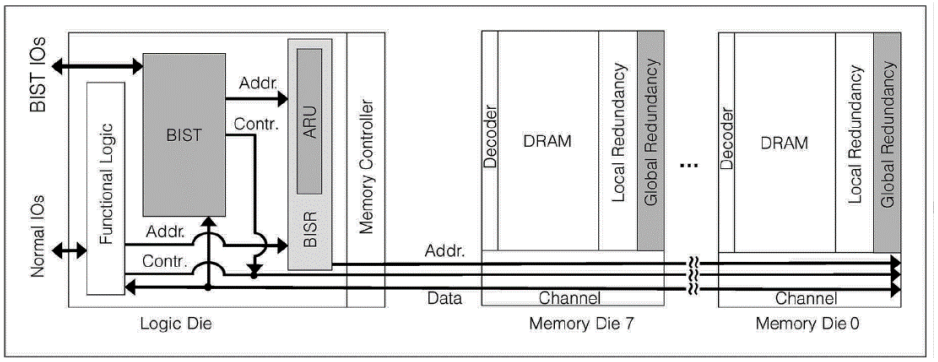


Figure 5. 3D memory repair scheme with redundancies in memory dies [16]

4.2. In-field repair

Conventionally redundancy has been added in memories to be used during production tests to correct defects and improve yield. However, these could be used to correct any soft defects seen in-field. Due to environmental conditions or aging, latent or hard errors may be introduced in memories during application. The in-field repair improves manufacturing yield as well as customer satisfaction. In case devices such as phones or tablet gets dead, they could get a firmware or software update from customer service. This section summarizes methods to do the incremental repair for such errors.

Becker [18] presented a processor that corrects latent or hard errors during online testing. The processor incorporates a memory built-in self-test (MBIST) controller, which reads data from memory and its ECC code to determine soft errors. Then, it writes back the corrected value. It also has a feature to do SRAM repair by replacing faulty lines with redundant rows or columns. Information about the errors could be reported to error management software.

Querbach *et al.* [17] presented a repair architecture for field use. The repair feature can be automatic or manual and run at boot via BIOS and firmware. Another method of correcting hard and soft errors during in-field operation is presented in [20]. It corrects: i) a single error, ii) two hard errors, and iii) one hard and one soft error using single error correction double error detection (SEC-DED) ECC.

A design-for test (DFT) architecture supporting memory repair during power-on and the in-system test is presented in [19]. At the start of power on, repair data from the manufacturing stage is loaded from one-time programmable (OTP) memory in BISR chains. The OTP controller works with power-on self-test (POST) controller and test-access port (TAP) controller to achieve this. Then the MBIST controller checks memories by running specified algorithms. If any new faults are detected, they are corrected using the soft repair technique presented in [59]. The time taken for loading data from OTP is critical as it adds to boot time. In conventional architecture, a single chain connects individual blocks in series. The author suggests having multiple BISR controllers and building separate memory configuration chains, as shown in Figure 6. The chains could be loaded in parallel to decrease loading time. Further, the author suggests storing repair data in the fuse in an uncompressed format to be loaded in the chain directly.

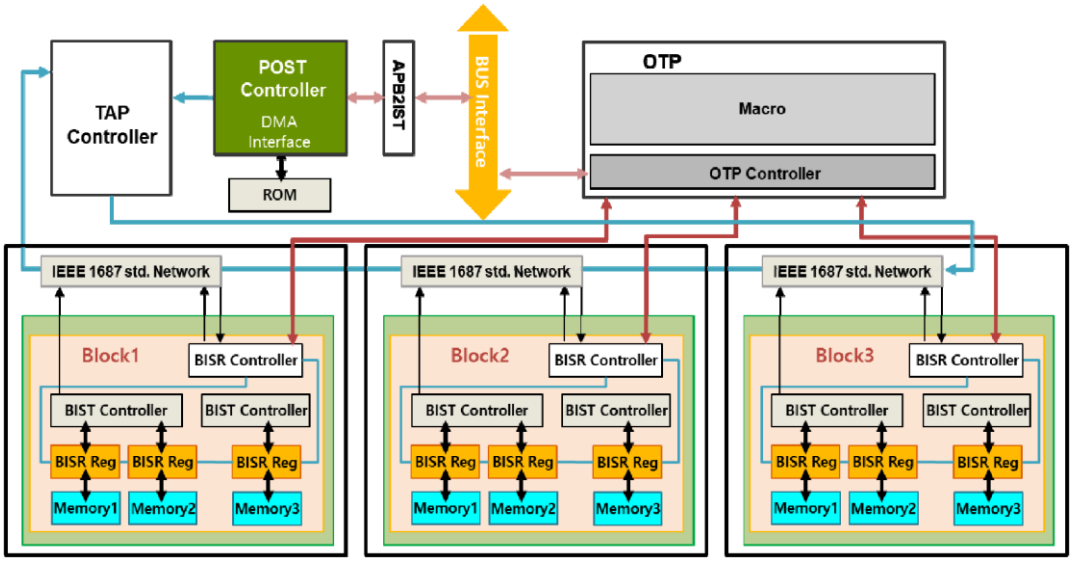


Figure 6. BISR Architecture supporting repair in-field [19]

4.3. Error correction code usage

Soft errors may be introduced in a system due to environmental conditions' variation or exposure to radioactive and alpha particles [2]. Memories are prone to such errors, and their probability increases with the size of the memories. Error correction codes have been widely used to correct these errors, thereby improving the yield and reliability of memories. Hamming and Hsiao codes correct one error and detect two errors. They are simple to implement, and encoding and decoding operations can be performed with them at high-speed [60]. This section summarizes these techniques.

Figure 7 shows a BISR scheme [20] that uses SEC-DED code-based ECC in addition to spare memory. When BIST detects faults, the information is sent to the BIRA module. BIRA module does not work on single-cell faults and leaves them to be corrected using ECC. It executes the ESP RA algorithm [6] to allocate spares for memory rows/columns containing multiple faults. This segregation significantly reduces the RA algorithm's complexity and achieves a high repair rate. During power-on reset, ECC is used to detect and correct single-cell faults. The author has shown results for maintaining reliability even though ECC is used for correcting single permanent faults.

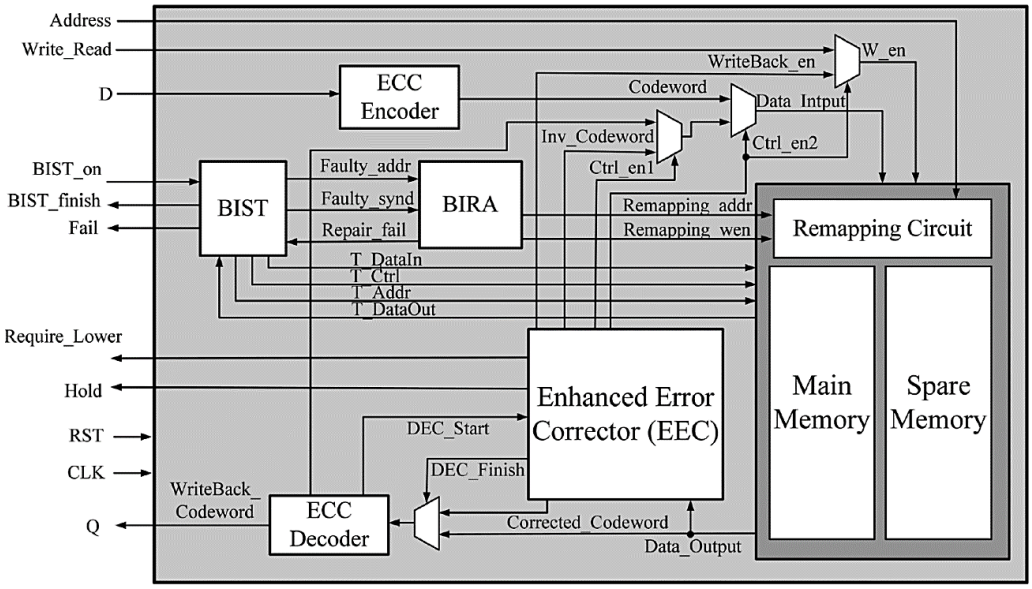


Figure 7. Architecture implementing both repair and ECC for memory faults [20]

Ganapathy *et al.* [25] also used SEC-DEC-based ECC to correct errors resulting from low voltage (LV) operation, which is being performed to improve power consumption. Prior schemes presented in [61] and [62] require additional MBIST runs when detected errors exceed the capacity of implemented code. This additional step increases boot time and delays the transition to the desired low-voltage state. The scheme presented in [25] improves this by dynamically detecting and correcting errors instead of using MBIST. In addition, it decouples ECC storage from the main cache, which enables it to scale it independent of cache size. The correction scheme presented in [63] uses decimal matrix code (DMC). It implements an encoder and decoder based on it on the data paths for writing and reading.

4.4. CAM usage for repair analysis

When BIST detects a new fault, its information is stored by BIRA in fault collection logic. The collection process must compare the incoming fault's address to the stored one. After BIST finishes the test, BIRA searches for the solution by accessing the stored address information. CAM is a particular type of memory that implements a lookup table, and comparison operations can be performed within a single cycle [64]. So, using CAM for storing fault information is preferred over other memory types like SRAM.

Cho *et al.* [26] presented a BIRA scheme that uses CAM. It categorizes a new fault as a pivot if it does not share a row and column with already identified faults. Figure 8 shows a few examples of such faults. Pivot faults play a crucial role in RA as spares allocation to them covers most of the repair solution. All non-pivot faults are mainly used to assist decisions made for pivot faults. The author uses CAM to store only pivot faults, unlike other RA schemes [10], which store both pivot and non-pivot faults in CAMs. All non-pivot faults are stored in spare memory. This storage scheme helps to reduce area overhead as CAM occupies more space than RAM.

The repair algorithm presented in [14] classifies faults as parent fault and child fault. A fault is classified as a child if it shares a row or column address with a parent fault. The fault collection process of the algorithm uses separate CAMs to store information on these faults.

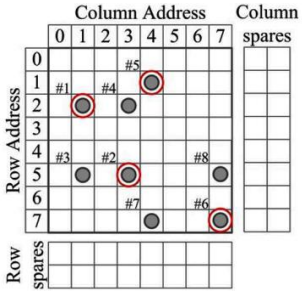


Figure 8. Pivot and non-pivot faults [26]

4.5. SRAM usage for repair analysis

BIRA requires storage to save information about faulty addresses. Many architectures proposed to use of CAM for this purpose. However, the usage of CAM adds to the area overhead. Also, since a single CAM is usually used due to its bigger size, it is shared. The sharing limits analysis speed as memories need to be analyzed sequentially. Few other architectures use a bank of SRAM as a bitmap for storing fault information. However, this results in too many accesses to memory and limits analysis speed as single storage is used and needs to be accessed sequentially.

Oh *et al.* [32] presented an architecture that uses a portion of its own memory for storing fault information. It first finds fault-free regions in the memory and then uses them for storing fault information. This architecture supports concurrent analysis on every memory. The author presented results that show analysis time is better than the branch-based algorithm (named as BRANCH) and almost similar to the address-mapping table (AMT) algorithm. Furthermore, the normalized repair rate is almost identical for all three algorithms. To note, the BRANCH [65] and AMT [66] algorithms use CAMs, which add area overhead compared to the method suggested by the author. The author later presented another architecture [33] that used fault-free regions from its own memory and showed to achieve optimal repair rate with a small area overhead.

Hou *et al.* [34] presented a scheme that uses a separate memory for storing fault information. This memory is used only in test mode. The author further modifies it as spare memory in normal mode. The scheme improves the repair rate by up to 11.95% as more resources are available for replacing faulty locations, while the hardware overhead to modify the memory is only 0.44%.

4.6. DRAM usage for repair analysis

Park *et al.* [37] used storage in DRAM itself for storing fault information. It also uses the internal BIRA feature in DRAM. These two features help to reduce area overhead for implementing repair solutions.

4.7. Using separate memory for redundancy instead of spare rows/columns

BIST is used to identify any failures in memory, and BISR implements repair functionality using fuses burned on ATE. However, this methodology of burning fuses is not available to the end user. Another way to achieve a repair capability is to add spare memory, which can be used to replace faulty bits. This section discusses such architectures.

Ryabtsev *et al.* [42], [44] presented architectures that use a backup memory to repair main memory faults. Based on susceptibility to failures, a small-size memory compared to the main memory is used for backup. Figure 9 shows the architecture consisting of 4-bit wide memory for backup, whereas the main memory is 64 bits wide. BISR sends reconfiguration code to the input data bus for writing in backup memory. Data from backup memory is used instead of the main memory for faulty locations in the main memory. Earlier, the author presented similar architecture [45], which uses a spare array to repair faults in critical applications systems like nuclear plants.

Architectures presented in [38], [39], [41], [43], [46] also used separate memory to implement redundancy to correct faults in main memories. Liu *et al.* [40] use a different approach to improve the system's robustness when encountering stuck faults on embedded SRAMs. Upon finding a faulty location, the system software rearranges data for writing. A barrel shifter is implemented in hardware that shifts data from memory and presents the corrected data to the system while reading the memory.

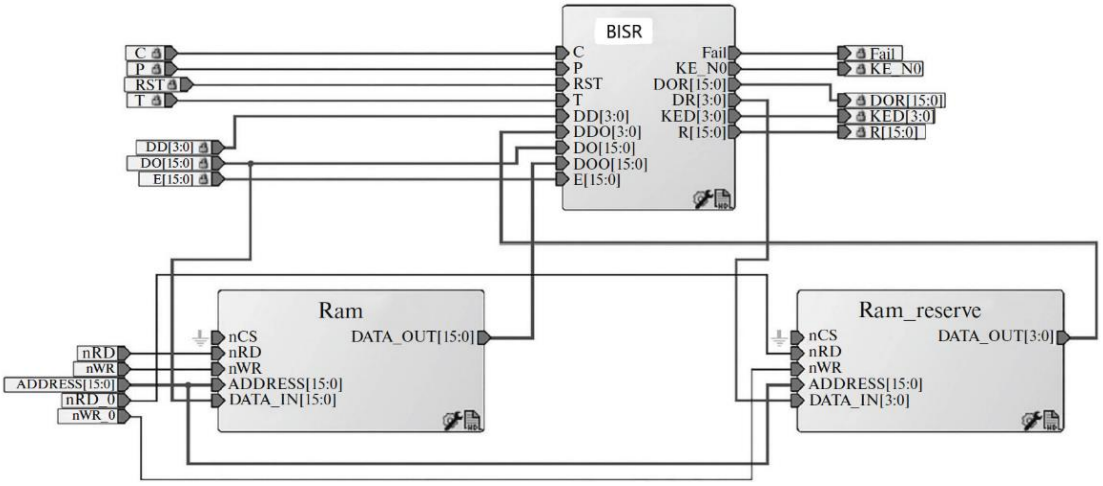


Figure 9. Repair architecture with spare memory for redundancy [42]

4.8. Standardization approaches

IEEE standard 1450.6.2-2014 provides guidelines to create memory models which could be used while designing MBIST-based test systems, validation, and failure analysis. Arora *et al.* [48] discussed the characteristics of utilizing these guidelines. The standard describes redundancy features, their enablement, mapping logical to physical addresses, and error injection. Memory IP providers could use these guidelines while developing memory models. SOC developers will have consistent memory models if they are sourced from different providers, and the providers follow the guidelines. The paper also points out opens in current standard guidelines like modeling for memory with different banks, limitations of scrambling sections, and few syntax difficulties.

The bitmapping procedure sends out information about faults outside the chip using the MBIST controller. A diagnostic step is required to translate the address into physical locations on memory to localize the defect. Un-optimized test settings or scrambling information inaccuracies may result in wrong localization. Study [49] presents details of stop-on-nth-error (SOE) and enhanced stop-on-nth-error (ESOE) flows used for creating bitmaps and finding physical coordinates of failing memory locations. Accurate diagnosis of defect location is vital to successful physical failure analysis as this further helps to understand the manufacturing

problem leading to those defects. The paper presented possible problem areas while doing bit mapping. It further provides a checklist of items in the test setup for a proper bit-mapping procedure.

High-performance IPs use a portion of functional datapath for MBIST purposes. IPs may use different implementations for the datapaths. Devanathan *et al.* [47] presented an architecture that supports the testing and repairing of such IPs with the diverse implementation of BIST datapaths. A separate RA block is used for each datapath implementation in conventional architecture. However, it leads to significant area overhead. In this paper, the author proposes to have single BIRA logic. For its implementation, it uses a uniform format for incoming repair data and fuse code to be downloaded in memory configuration registers. It aligns the failed signatures from each memory. Padding with 0 is applied for the non-existing repair part. A fuse and shadow reconfiguration block transforms the unified formatted fuse code to the target memory configuration register.

4.9. ROM error corrections

One-time programmable (OTP) ROM are used in many electronic systems like computers and mobile phones. They store user and system configuration information. Memory repair configuration information is also stored in eFuse-based ROM. They consist of programming transistors and fuse links. Since they consist of metal links, they are prone to age defects. This section summarizes techniques to detect and correct the defects in such ROMs.

Repair signatures of faulty memories are stored in eFuse, and the signature is read every time before beginning normal operation. Since eFuse plays a critical role, redundancy up to 2X is implemented conventionally. An ECC scheme for such OTP memory is presented in [22]. It implements Hamming code which corrects one error and detects two errors. Checksums for different memory regions are calculated and fed to an engine that also receives initially computed checksums. By comparing checksums, the engine detects errors in data. Read data from eFuse is stored in a temporary register, and ECC correction is applied to it. Corrected data is sent for actual usage. The scheme reduced the area of the ROM chip from 0.19 to 0.17 mm^2 compared to shadow register implementation with 2X redundancy. Since shadow register is not used, the time required to write them is also eliminated, which saves tester time by around 1,000 ms.

Birla and Costa [23] presented another ECC correction scheme for eFuse. The author adds encoder and decoder blocks on the written and read data, respectively, as shown in Figure 10. The BISR controller is modified to interface with these blocks. Hamming code generates a 6-bit checksum and is stored as additional MSB bits in eFuse. While reading data, the checksum is compared, and data is corrected for any faults. The area overhead of the scheme is 10% compared to 50% in conventional methods that uses 2X redundancy.

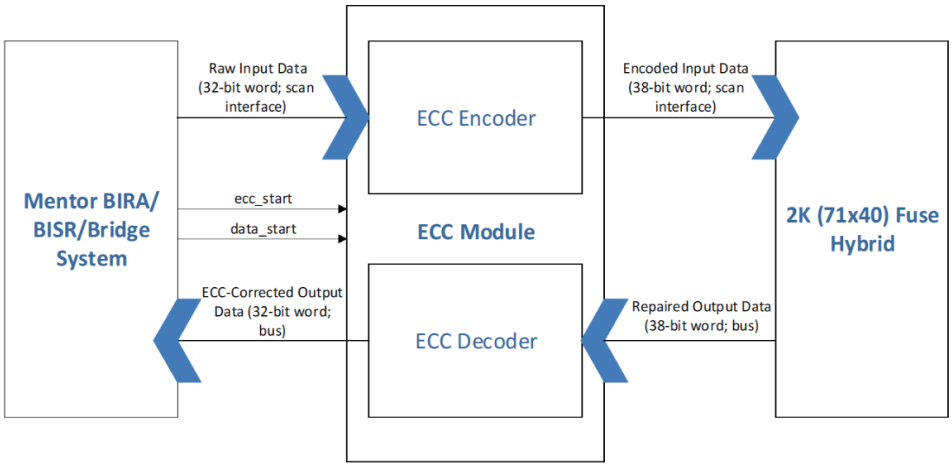


Figure 10. ECC-based correction scheme for fuse memory faults [23]

4.10. Hardware methods for defect localization

MBIST is popularly used for debugging memory failures. Though the bit-mapping method implemented using MBIST is software-based, implementation requires much effort. One of the challenges lies in getting logical to physical mapping information for memories. SOC consists of IPs that are sourced from multiple places. A few times, this mapping information is not available. Hardware-based failure analysis methods serve as an alternative in such situations.

Yeoh *et al.* [50] presented photon emission microscopy (PEM) based analysis to debug memory failures. PEM technique can detect defects caused by leakage or shorts [3]. These defects emit photons and create hotspots. These hotspots could be analyzed further to locate the defect. The analysis starts by checking the modulation of current at voltage drain to drain (VDD) supply at each cycle of MBIST test execution. Cycles with input-distribution device (IDD) current changes represent abnormal leakages. Such cycles are analyzed to know which memory or MBIST controller was active during those cycles. These memories become a region of interest for PEM analysis. A limitation of this method is its ineffectiveness for ohmic faults.

A laser heats the voids resulting in the paths becoming faster. This technique could be used to analyze speed path-related faults. Yeoh *et al.* [51] presented a laser-assisted device alteration (LADA) methodology to localize soft and hard failures in memory. Using LADA methodology, the author presented a case study to show memory fail-to-pass region. This methodology could be used as a complement to bit-mapping and PEM for memory failure debugging.

5. REPAIR ANALYSIS TYPES

5.1. Dynamic repair analysis

Most conventional RA algorithms start searching for solutions for allocating redundancy elements after memory testing is finished and information about all faults is available. Lee *et al.* [28] presented a dynamic built-in repair analysis algorithm that provides a repair solution as soon as BIST finishes executing testing. It does not wait to start finding repair solutions till BIST finishes and all fault addresses are available. It uses 4 CAMs for storing fault information – row main, row sub, column main, and column sub. Row-sub and column-sub store information about shared spares. The algorithm analyzes incoming faults and stores must-repair faults in main CAMs and other faults in sub-CAMs. It dynamically checks possible spares combinations to find solutions for currently detected faults. It moves fault addresses from sub-CAMs to main CAMs and vice-versa. As BIST finishes testing, addresses in main CAMs provide the final repair solution. The algorithm achieves an optimal repair rate without additional repair analysis time.

Kim *et al.* [52] presented a graphics processing unit (GPU) based dynamic RA technique. It analyses memory faults immediately instead of storing them. It uses GPU for parallel computation and analyses multiple RA computations efficiently. The normalized repair rate achieved using it is 100%, and the speed of analysis is significantly high.

5.2. Off-chip repair analysis

Kim *et al.* [9] proposed an ATE-based RA algorithm that reduces analysis time. It uses multiple types of spares, including local, common, and global. The global spare can be used to repair faults on the same row address in multiple banks simultaneously, which reduces analysis time. In addition, the common spares could be used to repair faults in adjacent banks instead of just one bank, which is the case with local spares. These two flexibilities help to achieve a very high repair rate. To reduce the storage space required for fault information on ATE, the algorithm store fault information as four lists: fault, pivot, child, and global spares. This pre-solution further helps to reduce analysis time. The algorithm also uses early termination effectively. RA analysis improvements of this algorithm over branch-and-bound (B&B) [67], very efficient redundancy analysis (VERA) [68], and fault group pattern matching (FGPM) [53] algorithms are as follows:

$$Repair\ rate: 0.6818(B\&B), 0.9470(VERA), 0.9470(FGPM), 0.9988(proposed)$$
$$Analysis\ time: 3.6440(B\&B), 1.2612(VERA), 0.9067(FGPM), 0.5276(proposed)$$

Another off-chip RA analysis scheme is presented in [52]. It performs RA analysis simultaneously as memory test operations are going on. The analysis results are available as soon as the memory test finishes. The author showed that RA latency using it is better than B&B, VERA, FGPM, repair most (RM) [69], and algorithm in [70].

5.3. Early termination repair analysis

Repair analysis algorithms work towards finding solutions for all faults using combinations of available spares. Finding this solution is an NP-complete problem. Identifying memories that cannot be repaired saves the remaining analysis time. A fault group pattern matching-based algorithm is presented in [53], which performs RA operation at high-speed using early termination. The previous algorithm proposed by the author with early termination [70] requires improvement as it depends on the number of spare elements in memory and fault population. In [53], groups of faults are formed which have the same row and same columns. The early termination algorithm first calculates the minimum spares required for each fault group. It then calculates the number of spares available after must-repair allocation. Early termination happens when the sum

of minimum spares for all fault groups exceeds the available spares. The author compared the results obtained with the algorithm with the VERA and local repair-most (LRM) [6] algorithms and showed the effectiveness of the proposed algorithm.

A similar fault grouping-based early termination is used in [9]. It creates groups of faults having the same row or column. Early termination is performed if the number of groups for memory is more than the maximum available spares.

5.4. Improving repair operation clock frequency

Repairing techniques based on redundant elements in memory requires replacing faulty lines with redundant lines. The most widely used method is through bit shifting using a control switch [71]. It compares a column address with shift point registers, and due to this, the path becomes critical, reducing memory access latency. Further, the latency suffers severely when data width increases as the delay is linearly proportional to the number of bits. Choi *et al.* [54] presented an architecture to reduce access latency by employing a parallel approach. It uses an encoder and decoder for arranging data to memory as a separate unit, as shown in Figure 11. The address comparison operation is done in parallel for each bit line. Therefore, all bits are encoded concurrently, unlike serial operation. With this approach, a critical path is determined by the number of redundant elements and not by the width of a data bus.

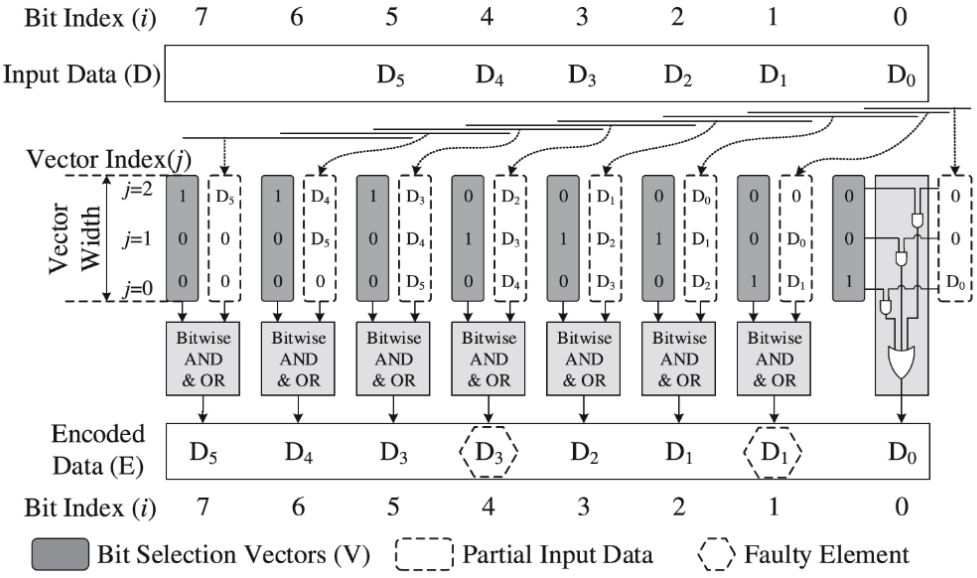


Figure 11. Parallel approach for faulty element replacement with redundant element [54]

Im *et al.* [19] proposed using multiple BISR controllers and accessing fuse data in parallel. This approach helps in 2 ways. i) Perform the loading of repair data in BISR chains in parallel to reduce test time and improve boot time and ii) Incorporating a local BISR controller in the block removes top-level timing dependency, and timing closure could be done with higher clock frequency, increasing repair operation speed.

6. SIMULATORS FOR MEASURING RELIABILITY AND REPAIR EFFICIENCY

Lee *et al.* [55] presented a methodology to generate a set of memory configurations that can be used to compare the efficiency of various redundancy analysis algorithms. It considers memory models with various redundancy structures. It uses the Polya-Eggenberger fault distribution model, which is considered to be the most realistic. Different probability values are used for faults in a cluster, on the row lines and column lines. The author presented repair rate and time for various RA algorithms with different models generated using the methodology.

Atishay *et al.* [57] presented a statistical error and redundancy analysis simulator which generates faults similar to the manufacturing environment for DRAM. Most other fault simulators, including [55], generate defects randomly or use Binomal or Polya-Eggenberger distribution model, which does not represent defects on wafers. These simulators also lack in considering defects from evolving DRAM technologies. The proposed defect generator considers distribution at each level - wafer lot, the wafer, memory bank, row/column.

The paper presented simulator results with different configurations of wafer lot and chips and compared repair rates for different algorithms.

Scionti *et al.* [58] presented a simulation environment called SIERRA for evaluating memory redundancy algorithms. It evaluates algorithms for repair capability, power consumption during its execution, and area overhead. It considers memory with different fault configurations, sizes, and densities. It generates memory configurations based on realistic defect distribution and fault models. It considers the possibility of defects in spare elements as well. The environment provides an interface for external tools like fault simulators and memory models. The environment exports repair efficiency results to be presented in a graphical way for easier understanding.

Nair *et al*. [56] presented a fault simulator tool called FaultSim for evaluating reliability mechanisms' effectiveness in two-dimensional (2D) and three-dimensional (3D) stacked memories. The FaultSim tool can be used to evaluate various ECC and spare mechanisms for memories and TSVs. Inputs given to it are memory organization, fault models, ECC/repair scheme, and scrubbing scheme, as shown in Figure 12. It uses an event-based fault injection framework, determining the time between faults. This approach reduces simulation time by 5000X compared to an interval-based fault injection scheme. For example, the tool can simulate ChipKill, and Bose–Chaudhuri–Hocquenghem (BCH-1) codes 1 million times in only 33 and 34 seconds, respectively.

A methodology to estimate yield considering errors due to aging is presented in [5]. The method does not consider wafer and package level failure. It considers only aging-related failures caused by wear-out mechanisms. It uses Weibull distribution for aging-related errors. Memory density, number of banks, and number of rows and columns are considered for defining memory configurations. It also considers repair mechanism information, like the number of spare columns and rows. Monte Carlo simulations are performed with these configurations to generate bitmaps. Repair analysis algorithms are run on these bitmaps to estimate yield. The simulation results contain information about expected probability; based on this, the method estimates the memory's lifetime.

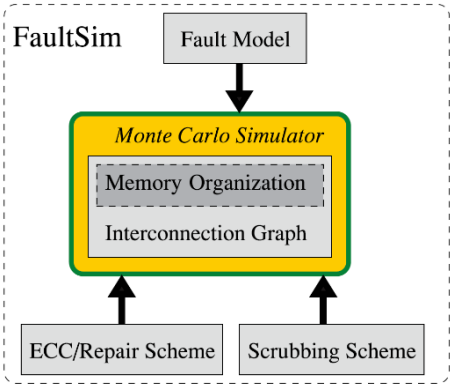


Figure 12. Components of fault simulator presented in [56]

7. DISCUSSION

After reviewing the recent memory repair technologies, this section discusses the benefits and main challenges that remain in its key areas. It also gives a direction for future work. Finally, a comparison of various repair algorithms is shown.

- 3D memories repair: improving yield is one of the major concerns for 3D memories. The yield depends upon the repair rates of individual 2D memories. If one of the memory dies is not repaired, the whole 3D package containing other good dies goes to waste. Therefore, the redundancy structure and RA algorithm need to be chosen carefully so that a high repair rate can be achieved both at the individual die level and package level. Sharing redundancies across dies and using global spares could be a way to achieve it.
- In-field repair: soft errors due to aging or environmental conditions must be corrected for automotive, safety-critical, and enterprise server applications. These applications necessitate having an incremental repair in the field that too periodically while running the application. Critical in-system tests to ensure all memories are fault-free are highly recommended for automotive and safety usage. Combinations of hard repair and ECC could be used. DFT architecture to control memory test and repair operations must work with the boot algorithm to support in-system tests and repair operations periodically.

- CAM/RAM/DRAM usage for RA: While CAM offers short access time, its area overhead is higher than RAM and DRAM. Several proposed architectures use shared CAM, their own RAM/DRAM, or separate RAM for fault information storage. The storage may demand a significant size based on the number of memory instances. The architectures must also be evaluated for implementation flow support in SOC.
- Simulators for measuring reliability and repair efficiency: a thorough evaluation of the RA algorithm gives insight to both the algorithm designer and the memory designer. Based on the results, changes could be made to improve yield further. Simulators like those presented in [57] and [58] could be effectively used for this purpose. Similarly, methodologies presented to estimate the memory lifetime could be used as an addition. These simulators together give insight for improvement before actually manufacturing the chip. Therefore, they need to be part of the memory repair methodology.
- Hardware methods for defect localization: for nanometer technologies, complex failure modes have become common. They will also be seen in upcoming technologies. Hardware-based techniques such as PEM [50] and LADA [51] could be used to complement the BISR-based hypothesis. These two techniques together could be used to finalize localization before the start of physical defect analysis. Such an approach will lead to increased accuracy and faster yield improvement.
- Using separate memory for redundancy instead of spare rows/columns: using redundancy based on spare rows and columns in memory itself requires using fuse macro, which is programmed on ATE. Such a fuse programming option may not be available for a few scenarios, including in-field usage. Using another memory or portion of available memory could be an option in such scenarios. Various architectures that use memory as a spare element are presented and could be adopted as needed.
- Standardization approach: multiple standards have been introduced to provide guidelines related to memory repair. First, IEEE 1450.6.2-2014 standard describes creating a memory model describing functional and repair operation. Then, ISO 26262 standard defines safety requirements for the automotive industry, motivating the need for in-field memory testing and incremental repair.

In addition to the repair ability of individual memory dies, interconnects in the form of tiny TSVs create a challenge in achieving high yield for the 3D package. All three components-logic die, memory dies, and interconnects- must be checked and repaired post-bonding. As a part of the wide input output (WIO) industry standard, JEDEC adopted a repair capability known as PPR.

7.1. Comparison of repair algorithm efficiency

Repair rate, area overhead, and analysis time are the main factors of the built-in repair analysis algorithm. An ideal RA algorithm must have a high repair rate, low area overhead, and short analysis time. However, spare allocation is an NP-complete problem [8], making achieving these three factors simultaneously difficult. The repair rate is considered the most important factor since it directly affects discarding repairable memory, leading to yield loss. Reducing storage required for fault information to reduce area overhead affects fault collection capability. Chips with bigger-size memories demand parallel testing and repair analysis to reduce test time. Table 5 summarizes the efficiency factor of various RA algorithms reviewed in this paper. The efficiency factor indicates the main goal achieved by the algorithm or implemented the feature in it.

Table 5. Repair algorithm efficiencies

Sr.	Efficiency Factor	Repair Algorithm
1	Area overhead efficient	[11], [23], [29], [31]–[33], [37], [46], [47], [72], [73]
2	Repair rate efficient	[15], [16], [20], [26], [34], [35], [74]
3	Analysis time efficient	[10], [28], [53], [54]
4	Area overhead and repair rate efficient	[25], [27], [36], [63]
5	Area overhead and analysis time efficient	[14], [22], [30], [37]
6	Repair rate and analysis time efficient	[9], [52]

A comparison of the repair rate and hardware overhead of BISR with different spares is presented in [63]. Memory banks with different configurations of local, common, and global spares are used for analysis. It showed that if memory has multiple types of spares and their combinations are large in number, a bigger size repair analyzer is required. Further, it indicates that common spares are most helpful in increasing repair rates.

8. CONCLUSION

With shrinking technologies, memory failure rates are going to increase. The failures will make memories an even more significant contributor to chip yield loss. Further, 3D memories will keep bringing a challenge to test new failures in TSVs from the efforts to use smaller size packages. These things demand exploring methodologies for improving fault-free memory operation. This paper presented a survey of repair

using redundancy and error-correcting code-based techniques used to handle hard and soft errors. It also presented a brief on various repair algorithms and simulators to evaluate their efficiency. Finally, it presented a comparison of repair algorithms and outlined challenges to be addressed in the methodologies.

REFERENCES

[1] E. I. Vatajelu, P. Prinetto, M. Taouil, and S. Hamdioui, "Challenges and solutions in emerging memory testing," *IEEE Transactions on Emerging Topics in Computing*, vol. 7, no. 3, pp. 493–506, Jul. 2019, doi: 10.1109/TETC.2017.2691263.

[2] R. Baumann, "Soft errors in advanced computer systems," *IEEE Design and Test of Computers*, vol. 22, no. 3, pp. 258–266, May 2005, doi: 10.1109/MDT.2005.69.

[3] M. R. Bruce and V. J. Bruce, "ABCs of photon emission Microscopy," *EDFA Technical Articles*, vol. 5, no. 3, pp. 13–20, Aug. 2003, doi: 10.31399/asm.edfa.2003-3.p013.

[4] S. Lee, K. Kim, Y. Lee, E. Lee, Y. Kim, and I. Kapilevich, "Marginal failure diagnosed with LADA—case studies," in *Conference Proceedings from the International Symposium for Testing and Failure Analysis*, Nov. 2014, pp. 358–364, doi: 10.31399/asm.cp.istfa2014p0358.

[5] D.-H. Kim and L. S. Milor, "Memory yield and lifetime estimation considering aging errors," in *2015 IEEE International Integrated Reliability Workshop (IIRW)*, Oct. 2015, pp. 130–133, doi: 10.1109/IIRW.2015.7437085.

[6] C.-T. Huang, C.-F. Wu, J.-F. Li, and C.-W. Wu, "Built-in redundancy analysis for memory yield improvement," *IEEE Transactions on Reliability*, vol. 52, no. 4, pp. 386–399, Dec. 2003, doi: 10.1109/TR.2003.821925.

[7] C.-T. Huang, J.-R. Huang, C.-F. Wu, C.-W. Wu, and T.-Y. Chang, "A programmable BIST core for embedded DRAM," *IEEE Design and Test of Computers*, vol. 16, no. 1, pp. 59–70, 1999, doi: 10.1109/54.748806.

[8] J. Day, "A fault-driven, comprehensive redundancy algorithm," *IEEE Design & Test of Computers*, vol. 2, no. 3, pp. 35–44, Jun. 1985, doi: 10.1109/MDT.1985.294737.

[9] H. Kim, H. Lee, D. Han, and S. Kang, "Multibank optimized redundancy analysis using efficient fault collection," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 8, pp. 2739–2752, Aug. 2022, doi: 10.1109/TCAD.2021.3109859.

[10] H. Lee, J. Kim, K. Cho, and S. Kang, "Fast built-in redundancy analysis based on sequential spare line allocation," *IEEE Transactions on Reliability*, vol. 67, no. 1, pp. 264–273, Mar. 2018, doi: 10.1109/TR.2017.2778301.

[11] J. Kim, W. Lee, K. Cho, and S. Kang, "Hardware-efficient built-in redundancy analysis for memory with various spares," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 3, pp. 844–856, Mar. 2017, doi: 10.1109/TVLSI.2016.2606499.

[12] G. P. Acharya and M. A. Rani, "Survey of test strategies for system-on chip and it's embedded memories," in *2013 IEEE Recent Advances in Intelligent Computational Systems (RAICS)*, Dec. 2013, pp. 199–204, doi: 10.1109/RAICS.2013.6745473.

[13] K. Cho, W. Kang, H. Cho, C. Lee, and S. Kang, "A survey of repair analysis algorithms for memories," *ACM Computing Surveys*, vol. 49, no. 3, pp. 1–41, Sep. 2017, doi: 10.1145/2971481.

[14] W. Kang, C. Lee, H. Lim, and S. Kang, "A 3 dimensional built-in self-repair scheme for yield improvement of 3 dimensional memories," *IEEE Transactions on Reliability*, vol. 64, no. 2, pp. 586–595, Jun. 2015, doi: 10.1109/TR.2015.2410274.

[15] H.-H. Liu *et al.*, "A built-off self-repair scheme for channel-based 3D memories," *IEEE Transactions on Computers*, vol. 66, no. 8, pp. 1293–1301, Aug. 2017, doi: 10.1109/TC.2017.2667645.

[16] B.-Y. Lin *et al.*, "Configurable cubical redundancy schemes for channel-Based 3-D DRAM yield improvement," *IEEE Design and Test*, vol. 33, no. 2, pp. 30–39, Apr. 2016, doi: 10.1109/MDAT.2015.2455347.

[17] B. Querbach, R. Khanna, S. Puligundla, D. Blankenbeckler, P. Y. Chiang, and J. Crop, "Architecture of a reusable BIST engine for detection and autocorrection of memory failures and for IO debug, validation, link training, and power optimization on 14-nm SoC," *IEEE Design and Test*, vol. 33, no. 1, pp. 59–67, Feb. 2016, doi: 10.1109/MDAT.2015.2445053.

[18] A. Becker, "Short burst software transparent on-line MBIST," in *2016 IEEE 34th VLSI Test Symposium (VTS)*, Apr. 2016, pp. 1–6, doi: 10.1109/VTS.2016.7477287.

[19] S. Im, G. Nam, S. Park, and M. Noh, "Advanced safety test solution for automotive SoC based on in-system-test architecture," in *2022 IEEE International Symposium on Circuits and Systems (ISCAS)*, May 2022, pp. 2290–2293, doi: 10.1109/ISCAS48785.2022.9937235.

[20] S.-K. Lu, C.-J. Tsai, and M. Hashizume, "Enhanced built-in self-repair techniques for improving fabrication yield and reliability of embedded memories," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 24, no. 8, pp. 2726–2734, Aug. 2016, doi: 10.1109/TVLSI.2016.2523499.

[21] V. R. G., T. Y. Utkina, and M. Almadi, "Automatic operability restoration of semiconductor memory's modules during multiple faults," *European Journal of Engineering and Technology*, vol. 3, no. 5, pp. 72–79, 2015.

[22] H. Divva, A. P. Chavan, and S. Krishnamurthy, "Design and verification of ECC scheme to optimize area and tester time in OTP ROM controller," in *2019 4th IEEE International Conference on Recent Trends on Electronics, Information, Communication and Technology, RTEICT 2019 - Proceedings*, May 2019, pp. 151–155, doi: 10.1109/RTEICT46194.2019.9016746.

[23] S. Beerla and M. Costa, "Enabling ECC and repair features in an eFuse box for memory repair applications," in *2021 IEEE 39th VLSI Test Symposium (VTS)*, Apr. 2021, pp. 1–4, doi: 10.1109/VTS50974.2021.9441028.

[24] S. S. R. Dannina and U. V. R. Kumari, "Error tolerant MBIST design with efficient high throughput and low power reversible techniques," in *2021 10th IEEE International Conference on Communication Systems and Network Technologies (CSNT)*, Jun. 2021, pp. 230–235, doi: 10.1109/CSNT51715.2021.9509611.

[25] S. Ganapathy, J. Kalamatianos, B. M. Beckmann, S. Raasch, and L. G. Szafaryn, "Killi: runtime fault classification to deploy low voltage caches without MBIST," in *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, Feb. 2019, pp. 304–316, doi: 10.1109/HPCA.2019.00046.

[26] K. Cho, Y. W. Lee, S. Seo, and S. Kang, "An efficient BIRA utilizing characteristics of spare pivot faults," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 3, pp. 551–561, Mar. 2019, doi: 10.1109/TCAD.2018.2818725.

[27] G. Wang and C. Chang, "Design and implementation of shared BISR for RAMs: a case study," in *2016 IEEE AUTOTESTCON*, Sep. 2016, pp. 1–7, doi: 10.1109/AUTEST.2016.7589621.

[28] H. Lee, D. Han, S. Lee, and S. Kang, "Dynamic built-in redundancy analysis for memory repair," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 10, pp. 2365–2374, Oct. 2019, doi: 10.1109/TVLSI.2019.2920999.

[29] W. Lee, K. Cho, J. Kim, and S. Kang, "Near optimal repair rate built-in redundancy analysis with very small hardware overhead,"

in *Sixteenth International Symposium on Quality Electronic Design*, Mar. 2015, pp. 435–439, doi: 10.1109/ISQED.2015.7085465.
[30] W. Kang, C. Lee, H. Lim, and S. Kang, "Optimized built-in self-repair for multiple memories," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 24, no. 6, pp. 1–10, 2015, doi: 10.1109/TVLSI.2015.2499387.
[31] D. Kim, H. Lee, and S. Kang, "An area-efficient BIRA with 1-D spare segments," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 26, no. 1, pp. 206–210, Jan. 2018, doi: 10.1109/TVLSI.2017.2752298.
[32] C.-H. Oh, S.-E. Kim, and J.-S. Yang, "A BIRA using fault-free memory region for area reduction," in *2016 IEEE Asia Pacific Conference on Circuits and Systems (APCCAS)*, Oct. 2016, pp. 480–482, doi: 10.1109/APCCAS.2016.7804008.
[33] C.-H. Oh, S.-E. Kim, and J.-S. Yang, "BIRA with optimal repair rate using fault-free memory region for area reduction," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 64, no. 12, pp. 3160–3171, Dec. 2017, doi: 10.1109/TCSI.2017.2750702.
[34] C.-S. Hou and J.-F. Li, "High repair-efficiency BISR scheme for RAMs by reusing bitmap for bit redundancy," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 23, no. 9, pp. 1720–1728, Sep. 2015, doi: 10.1109/TVLSI.2014.2354378.
[35] A. K. S. Pundir, "Novel modified memory built in self-repair (MMBISR) for SRAM using hybrid redundancy-analysis technique," *IET Circuits, Devices and Systems*, vol. 13, no. 6, pp. 836–842, Aug. 2019, doi: 10.1049/iet-cds.2018.5218.
[36] S. Alnatheer and M. A. Ahmed, "Optimal method for test and repair memories using redundancy mechanism for SoC," *Micromachines*, vol. 12, no. 7, Jul. 2021, doi: 10.3390/mi12070811.
[37] J. Park, J. H. Lee, S.-K. Park, K. C. Chun, K. Sohn, and S. Kang, "An in-DRAM BIST for 16 Gb DDR4 DRAM in the 2nd 10-nm-Class DRAM Process," *IEEE Access*, vol. 9, pp. 33487–33497, 2021, doi: 10.1109/ACCESS.2021.3061349.
[38] K. H. Ng, N. E. Alias, A. Hamzah, M. L. P. Tan, U. U. Sheikh, and Y. A. Wahab, "A march 5n FSM-based memory built-in self-test (MBIST) architecture with diagnosis capabilities," in *2022 IEEE International Conference on Semiconductor Electronics (ICSE)*, Aug. 2022, pp. 69–72, doi: 10.1109/ICSE56004.2022.9863160.
[39] G. P. Acharya, M. A. Rani, G. G. Kumar, and L. Poluboyina, "Adaptation of of march-SS algorithm to word-oriented memory built-in self-test and repair," *Indonesian Journal of Electrical Engineering and Computer Science (IJEECS)*, vol. 26, no. 1, Apr. 2022, doi: 10.11591/ijeecs.v26.i1.pp96-104.
[40] S.-M. Liu, L. Tang, N.-C. Huang, D.-Y. Tsai, M.-X. Yang, and K.-C. Wu, "Fault-tolerance mechanism analysis on NVDLA-based design using open neural network compiler and quantization calibrator," in *2020 International Symposium on VLSI Design, Automation and Test (VLSI-DAT)*, Aug. 2020, pp. 1–3, doi: 10.1109/VLSI-DAT49148.2020.9196335.
[41] R. Manasa, R. Verma, and D. Koppad, "Implementation of BIST Technology using March-LR Algorithm," in *2019 4th International Conference on Recent Trends on Electronics, Information, Communication and Technology (RTEICT)*, May 2019, pp. 1208–1212, doi: 10.1109/RTEICT46194.2019.9016784.
[42] V. G. Ryabtsev and S. V Volobuev, "Built-in self-repairing system-on-chip RAM," *Russian Microelectronics*, vol. 50, no. 7, pp. 504–508, Dec. 2021, doi: 10.1134/S1063739721070118.
[43] A. S. Syed, M. A. Ahmed, and M. A. Ahmed, "Embedded memory test strategies and repair," *International Journal of Engineering*, vol. 30, no. 6, Jun. 2017, doi: 10.5829/ije.2017.30.06c.03.
[44] V. G. Ryabtsev and S. V Volobuev, "Implementation of memory in a system on a chip with built-in self-testing and self-healing," *Russian Microelectronics*, vol. 49, no. 7, pp. 527–531, Dec. 2020, doi: 10.1134/S1063739720070100.
[45] V. G. Ryabtsev, A. S. Feklistov, and K. V. Evseev, "Improved reliability memory's module structure for critical application systems," *International Journal of Engineering Research and Applications (IJERA)*, vol. 6, no. 1, pp. 65–68, 2016.
[46] M. A. Ahmed, A. E. M. Eljialy, and S. Ahmad, "Memory test and repair technique for SoC based devices," *IEICE Electronics Express*, vol. 18, no. 8, Apr. 2021, doi: 10.1587/elex.18.20210092.
[47] V. R. Devanathan and S. Kale, "A reconfigurable built-in memory self-repair architecture for heterogeneous cores with embedded BIST datapath," in *2016 IEEE International Test Conference (ITC)*, Nov. 2016, pp. 1–6, doi: 10.1109/TEST.2016.7805870.
[48] P. Arora, P. Gallagher, and S. L. Gregor, "Core test language based high quality memory testing and repair methodology," in *2021 IEEE International Test Conference India (ITC India)*, Jul. 2021, pp. 1–6, doi: 10.1109/ITCIndia52672.2021.9532722.
[49] L. Zhao *et al.*, "Accurate memory bitmapping based on built-in self-test: challenges and solutions," in *2019 IEEE 26th International Symposium on Physical and Failure Analysis of Integrated Circuits (IPFA)*, Jul. 2019, pp. 1–8, doi: 10.1109/IPFA47161.2019.8984869.
[50] B. L. Yeoh, S. H. Goh, G. F. You, H. Hao, W. L. Sio, and J. Lam, "Debugging MBIST hard fails without bitmapping," in *2015 IEEE 22nd International Symposium on the Physical and Failure Analysis of Integrated Circuits*, Jun. 2015, pp. 138–143, doi: 10.1109/IPFA.2015.7224352.
[51] B. L. Yeoh, M. H. Thor, L. S. Gan, Y. H. Chan, and S. H. Goh, "LADA methodologies to localize embedded memory failure," in *2022 IEEE International Symposium on the Physical and Failure Analysis of Integrated Circuits (IPFA)*, Jul. 2022, pp. 1–6, doi: 10.1109/IPFA55383.2022.9915715.
[52] T. H. Kim, H. Lee, and S. Kang, "GPU-based redundancy analysis using concurrent evaluation," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 28, no. 3, pp. 805–817, Mar. 2020, doi: 10.1109/TVLSI.2019.2954549.
[53] H. Lee, K. Cho, D. Kim, and S. Kang, "Fault group pattern matching with efficient early termination for high-speed redundancy analysis," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 7, pp. 1473–1482, Jul. 2018, doi: 10.1109/TCAD.2017.2760505.
[54] K. H. Choi, J. Jun, H. Kim, S. W. Kim, and Y. Han, "A decoupled bit shifting technique using data encoding/decoding for DRAM redundancy repair," *IEICE Electronics Express*, vol. 14, no. 13, 2017, doi: 10.1587/elex.14.20170385.
[55] H. Lee, K. Cho, S. Kang, W. Kang, S. Lee, and W. Jeong, "Fail memory configuration set for RA estimation," in *2020 IEEE International Test Conference (ITC)*, Nov. 2020, pp. 1–9, doi: 10.1109/ITC44778.2020.9325273.
[56] P. J. Nair, D. A. Roberts, and M. K. Qureshi, "FaultSim: a fast, configurable memory-reliability simulator for conventional and 3D-stacked systems," *ACM Transactions on Architecture and Code Optimization*, vol. 12, no. 4, pp. 1–24, Dec. 2015, doi: 10.1145/2831234.
[57] Atishay, A. Gupta, R. Sonawat, H. K. Thacker, and P. B., "SEARS: a statistical error and redundancy analysis simulator," in *2019 IFIP/IEEE 27th International Conference on Very Large Scale Integration (VLSI-SoC)*, Oct. 2019, pp. 117–122, doi: 10.1109/VLSI-SoC.2019.8920344.
[58] A. Scionti, S. Mazumdar, S. Di Carlo, and S. Hamdioui, "SIERRA—simulation environment for memory redundancy algorithms," *Simulation Modelling Practice and Theory*, vol. 69, pp. 14–30, Dec. 2016, doi: 10.1016/j.simpat.2016.08.008.
[59] B. Cowan, O. Farnsworth, P. Jakobsen, S. Oakland, M. R. Ouellette, and D. L. Wheater, "On-chip repair and an ATE independent fusing methodology," in *Proceedings. International Test Conference*, 2002, pp. 178–186, doi: 10.1109/TEST.2002.1041759.
[60] D. Rossi, N. Timoncini, M. Spica, and C. Metra, "Error correcting code analysis for cache memory high reliability and performance," in *2011 Design, Automation and Test in Europe*, Mar. 2011, pp. 1–6, doi: 10.1109/DATE.2011.5763257.
[61] C. Wilkerson, H. Gao, A. R. Alameldeen, Z. Chishti, M. Khellah, and S.-L. Lu, "Trading off cache capacity for reliability to enable low voltage operation," in *2008 International Symposium on Computer Architecture*, Jun. 2008, pp. 203–214, doi:

10.1109/ISCA.2008.22.
[62] M. K. Qureshi and Z. Chishti, "Operating SECDED-based caches at ultra-low voltage with FLAIR," in *2013 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, Jun. 2013, pp. 1–11, doi: 10.1109/DSN.2013.6575314.
[63] J. Kim, K. Cho, W. Lee, and S. Kang, "A new built-in redundancy analysis algorithm based on multiple memory blocks," in *2015 International SoC Design Conference (ISOCC)*, Nov. 2015, pp. 43–44, doi: 10.1109/ISOCC.2015.7401691.
[64] K. Pagiamtzis and A. Sheikholeslami, "Content-addressable memory (CAM) circuits and architectures: a tutorial and survey," *IEEE Journal of Solid-State Circuits*, vol. 41, no. 3, pp. 712–727, Mar. 2006, doi: 10.1109/JSSC.2005.864128.
[65] W. Jeong, J. Lee, T. Han, K. Lee, and S. Kang, "An advanced BIRA for memories with an optimal repair rate and fast analysis speed by using a branch analyzer," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 29, no. 12, pp. 2014–2026, Dec. 2010, doi: 10.1109/TCAD.2010.2062830.
[66] W. Kang, H. Cho, J. Lee, and S. Kang, "A BIRA for memories with an optimal repair rate using spare memories for area reduction," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 22, no. 11, pp. 2336–2349, Nov. 2014, doi: 10.1109/TVLSI.2013.2288637.
[67] S. Kuo and W. Fuchs, "Efficient spare allocation for reconfigurable arrays," *IEEE Design & Test of Computers*, vol. 4, no. 1, pp. 24–31, 1987, doi: 10.1109/MDT.1987.295111.
[68] H. Cho, "A very efficient redundancy analysis method using fault grouping," *ETRI Journal*, vol. 35, no. 3, pp. 439–447, Jun. 2013, doi: 10.4218/etrij.13.0112.0467.
[69] M. Tarr, D. Boudreau, and R. Murphy, "Defect analysis system speeds test and repair of redundant memories," *Microelectronics Reliability*, vol. 24, no. 4, Jan. 1984, doi: 10.1016/0026-2714(84)90243-9.
[70] H. Cho, W. Kang, and S. Kang, "A fast redundancy analysis algorithm in ATE for repairing faulty memories," *ETRI Journal*, vol. 34, no. 3, pp. 478–481, Jun. 2012, doi: 10.4218/etrij.12.0211.0378.
[71] T. Namekawa *et al.*, "Dynamically shift-switched dataline redundancy suitable for DRAM macro with wide data bus," in *1999 Symposium on VLSI Circuits. Digest of Papers (IEEE Cat. No.99CH36326)*, 1999, pp. 149–152, doi: 10.1109/VLSIC.1999.797267.
[72] M. A. Ahmed and S. Alnatheer, "Deep Q-learning with bit-swapping-based linear feedback shift register fostered built-in self-test and built-in self-repair for SRAM," *Micromachines*, vol. 13, no. 6, Jun. 2022, doi: 10.3390/mi13060971.
[73] K. Cho, Y. Lee, S. Seo, and S. Kang, "An efficient built-in self-repair scheme for area reduction," in *2017 International SoC Design Conference (ISOCC)*, Nov. 2017, pp. 105–106, doi: 10.1109/ISOCC.2017.8368791.
[74] J. Kim, K. Cho, W. Lee, and S. Kang, "A new redundancy analysis algorithm using one side pivot," in *2014 International SoC Design Conference (ISOCC)*, Nov. 2014, pp. 134–135, doi: 10.1109/ISOCC.2014.7087609.

BIOGRAPHIES OF AUTHORS

Vijay Sontakke received the B.E. degree in electronics engineering from Shri Guru Gobind Singhji Institute of Engineering and Technology, Nanded, India, in 1997. He received the M.Tech. degree in electronics engineering from Visvesvaraya National Institute of Technology, Nagpur, India, in 1999. He is currently a design engineer at Intel Corporation, Allentown, PA, USA. Before Intel, he worked at Motorola, AMD, Conexant, and Ikanos Communications. He possesses VLSI design experience focusing on the design-for-test. He worked for the entire life cycle of the design-for-test, from test specification definition to production test program release, defined architecture for several SOCs, and led implementation. His current research interests include test power minimization, pattern count optimization, JTAG/IJTAG, 3D IC testing, memory testing, and scan architecture. He can be contacted at email: vijay.sontakke@intel.com.

Delsikreo Atchina received a Bachelor of Science in Engineering from Messiah University, Grantham, USA, in 1998. He received the Master of Science in electrical engineering from Temple University, Philadelphia, USA, in 2003. He has been a Senior DFT Engineer at Intel Networking ASIC Group, Allentown, PA, USA, since 2014. He works for multimillion gate blocks and SOCs to integrate test structures to support scan, IJTAG, and memory testing. He also works for validation involving pattern generation, simulation, and test coverage analysis. Before Intel, he worked at Lucent, Agere, and LSI Corporation. Delsi also has extensive experience in timing closure and in place and route to address signal integrity issues and physical design rules for large SOCs. He can be contacted at email: delsi.atchina@intel.com.