Politecnico di Torino

ScuDo
Scuola di Dottorato ~ Doctoral School
WHAT YOU ARE, TAKES YOU FAR

Doctoral Dissertation
Doctoral Program in Computer and Control Engineering (37.th cycle)

Test and diagnosis of memories embedded in Automotive SoCs

Giorgio Insinga
* * * * *

Supervisors
Prof. P.Bernardi, Supervisor
Prof. R. Cantoro Co-supervisor

Politecnico di Torino
March 21, 2025

I hereby declare that, the contents and organisation of this dissertation constitute my own original work and does not compromise in any way the rights of third parties, including those relating to the security of personal data.

. . . . . .

Giorgio Insinga

Turin, March 21, 2025

Summary

This thesis addresses the critical challenge of ensuring the reliability and testability of embedded memories in Automotive Systems-on-Chips (SoCs), spanning from manufacturing to end-user reliability assessment. With the increasing complexity and safety-critical nature of automotive systems, coupled with shrinking technology nodes, ensuring the robust operation of embedded memories is paramount. This work presents novel methodologies for efficient diagnostic data collection, processing, and retrieval, along with hardware and software solutions for mitigating voltage droop issues and classifying fault shapes.

The starting point of this thesis is the deep characterization of the embedded memories of the Infineon Aurix SoCs produced by Infineon Technologies. In this step, embedded memories are tested under various condition of supply voltage, temperature, read timing etc.

After the characterization step, the challenge of efficiently manage the diagnostic data coming from the test of the embedded memories is tackled. Three on-chip diagnostic data collection methods are proposed and validated on an Infineon Aurix SoC. The first method employs "slices" for lossless failure bitmap encoding, achieving up to 99.8% diagnostic space savings compared to previous list-based methods. The second one further improves the first method by only reporting the differences between one test and the other, further saving 66.99% of diagnostic space. The third one utilizes "pixel slices" for density representation, offering up to 72.6% savings with respect to state-of-the-art lossless approaches, albeit with some loss of information. A novel hardware-software scheme for diagnostic data retrieval reduces test time by 25% and improves fault representation fairness in limited diagnostic memory scenarios. Furthermore, two low-area, low-complexity circuits are presented to mitigate voltage droop during testing, thereby enhancing diagnostic data reliability.

To facilitate advanced fault analysis, a ResNet18-based neural network is implemented to classify eight different fault shapes, outperforming existing approaches and enabling the recognition of novel fault patterns. Finally, this thesis presents software for reconstructing internal memory organization from irradiation test data, allowing end-users to simulate radiation effects and perform accurate reliability assessments. These contributions collectively enhance the testability and reliability of embedded memories in automotive SoCs, paving the way for more robust and dependable automotive systems.

Acknowledgements

First and foremost, I would like to thank my supervisors, Paolo Bernardi and Riccardo Cantoro, for their help and friendship during my PhD journey. Their knowledge and wisdom were invaluable in each issue encountered along the way. I would also like to acknowledge several people from Politecnico di Torino and Infineon Technologies who were incredibly helpful and supportive during my projects' brainstorming and development (and countless hours of debugging).

All the following people are mentioned in alphabetical order and are all from Politecnico di Torino:

- Prof. P. Bernardi
- M. Battilana
- F. Carpegna
- R. Cantoro
- A. M. Guerriero
- G. Insinga
- N. Kolahimahmoudi
- B. Borio
- B. Mendicino
- G. Paganini
- F. Sanna
- A. Ruospo

All the following people are mentioned in alphabetical order and are all distributed between Infineon Technologies Austria (Villach), Germany (Munich) and Italy (Padoa):

- P. Beer
- G. Carnevale
- M. Coppetta
- M. Giltrelli
- G. Koebernik
- N. Mautone
- J. Niederl
- S. Roggi
- P. Scaramuzza
- F. Tengler
- R. Ullmann
- W. Mischo
- Giambattista Carnevale
- Matteo Coppetta
- Nellina Mautone
- Josef Niederl
- Stefano Roggi
- Pierre Scaramuzza
- Rudolf Ullmann

A special thanks also to my PhD student colleague Nima Kolahimahmoudi, whose technical skills were invaluable during the project with Infineon Technologies Austria.

I would like to dedicate this thesis to my loving parents and grandparents. A special thank you also goes to my girlfriend Kiara for all the support she has given me during my PhD.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

This thesis addresses the increasingly critical challenge of ensuring the reliability and testability of memories embedded in Automotive System-on-Chips (SoC). This thesis provides a full overview of novel state-of-the-art approaches for memory tests and diagnosis, spawning from the manufacturing of the devices to the reliability assessment of the final users developing applications for the SoCs the eMemories are integrated into. Carefully studying this topic is paramount, as with technological advancements and with technology nodes constantly scaling down, new and unexpected fault mechanisms arise.

Technology is more pervasive than ever, especially in the automotive sector, where the number of electronic devices has grown exponentially in the last couple of decades. However, as SoCs become more and more complex with new peripherals and new capabilities, embedded memories (eMemories) also need to grow accordingly to satisfy the requirements of the new memory-hungry programs that take full advantage of modern microcontrollers. Consequently, eMemories end up occupying a significant percentage of the die area of each commercialized device. Their reliability deeply affects the yield and the quality of the produced device, and thoroughly testing them during production is paramount for modern SoC manufacturers. [1]

To address these multifaceted challenges, this thesis presents a comprehensive investigation into various aspects of testing and diagnosing memories embedded in Automotive SoCs to keep improving the state-of-the-art (SOTA) and make modern safety-critical devices as reliable as possible while also saving test costs for SoC manufacturers.

The key challenges addressed in this thesis include:

- Deep characterization of memories embedded in Automotive SoCs
- Efficient collection and processing of large volumes of diagnostic data
- Reduction of test time while ensuring fair representation of memory faults
- Mitigation of voltage droops during critical test phases

- Classification and interpretation of fault patterns using machine learning
- Reconstruction of internal memory organization for radiation effect simulation

The first step is the characterization process that Automotive SoC manufacturers perform on their eMemories to assess their characteristics before the products are launched on the market. During these tests, the manufacturers tests their eMemories in various conditions of temperature, reading speed, memory access time etc [1]. These extensive characterization procedures generate massive amounts of data that must be efficiently managed and analyzed.

Regarding diagnostic data management, this thesis presents three novel methods for efficiently collecting and processing diagnostic data. These methods are designed and validated to run on-chip on an Infineon Aurix Automotive-grade SoC and to reduce the amount of communication with the external testers to reduce transfer times.

1. The first method exploits the regular structure of the eMemories to perform an optimized fault shape recognition, resulting in a lossless collection of shape-encoding segments called "slices". Test performed on real Infineon Automotive SoCs shows that this approach saves up to 99.8% of the diagnostic space required by previous SOTA approaches [2].

2. The second method is an algorithm that expands the capabilities of the first one by encoding just the differences between a test and the next one. During the first test, the algorithms save the order of arrival of the faults in structures called delta slices. In this way during the second test, the algorithm can compute the next expected fault and immediately notice if there is a difference, if another fault is received. The algorithm uses colored slices to represent the differences found. This further saves an average of 66.99% diagnostic memory space with respect to the algorithm presented in the previous point.

3. The third method is oriented toward the density representation of failures in memory, reporting just the number of faults in a given square of memory but not their precise location in special structures called "Pixel slices". Even with their lossy nature, pixel slice results are incredibly valuable during some characterization steps as they allow test engineers to notice weaker areas of their memories immediately. Test performed on real Infineon Automotive SoCs shows that using this lossy approach, the total diagnostic memory save is up to 72.6% compared with the lossless slice approach.

While efficient data organization is crucial, the ability to retrieve and process this diagnostic information is equally important. Therefore, this thesis next addresses the challenge of effectively retrieving the diagnostic data to reduce test time and assess memory fairness in case of limited dedicated on-chip diagnostic

memory space. For these reasons also, the dedicated Memory Built-in-Self-Tests (MBISTs) [3][4] and how to efficiently manage them are addressed in this thesis. By implementing a novel hardware-software scheme in an Infineon Automotive SoC, a reduction of test time of 25% (with respect to previous linear-search-based methods [5]) is achieved while also having the added benefit of a fair representation of the faults in the memory.

Apart from managing the Design for Testability (DfT) hardware of the eMemories, other external factors need to be considered for a successful memory test such as:

- Supply voltage variation for characterization purposes
- Voltage droops caused by the sudden change in the current requirements
- Bit flips caused by cosmic radiations

For example, during characterization, the SoCs are tested at various ranges of supply voltages [1]. In particular, low voltages are often caused by issues during the test flow, as sudden current requirements create dangerous voltage droops [6] [7]. Two novel low-area and low-complexity circuits have been designed to solve dangerous voltage droops by temporarily raising the voltage in critical parts of the test flow. These hardware optimizations ensure reliable data collection, setting the stage for comprehensive failure analysis.

With robust data collection and hardware solutions in place, the next critical challenge is to classify and interpret the accumulated diagnostic data effectively[8][9][10]. To this end, this thesis presents a novel SOTA neural network based on the ResNet18 architecture to classify 8 different fault shapes. Apart from performing significantly better than previous SOTA approaches, the proposed solution allows for the recognition of new and unseen fault shapes to speed up the diagnosis of novel memory fault mechanisms.

Up until this point this thesis focused on what manufacturers can do to improve the tests and diagnosis of their eMemories. However, manufacturers are not the only entities interested in the reliability of their products.

In safety-critical environments such as automotive and aerospace, assessing the reliability of electronic systems is paramount. In these environments final SoCs programmers:

- need to test their application against the most common issues arising during their systems' operational life
- don't know the internal organization of the devices they are programming, including the memories
- wants to simulate the effect of various physical phenomena on the reliability of their systems

A classic example of a threat that affects electronic devices is cosmic radiation made of high-energy charged particles[11][12][13]. This phenomenon is dangerous as it can interact with transistors and flip their state, generating Single Event Upsets (SEUs) and Multiple Event Upsets (MEUs)[14]. Various countermeasures have been devised to mitigate this issue, such as Error Correction Code (ECC) and other correction mechanisms for the most critical components (duplication, triplication, etc).

Due to their regular matrix structures, having multiple bits near each other, eMemories are particularly prone to MEUs, which would flip multiple bits at once. This is particularly true for charged particles that strike at the intersection between two or more cells. But to study the effects of these radiation-induced errors on the execution of the programs stored in the affected memories, final users would need to simulate the effects of MEUs in their system. However, having a simulation of the eMemories is a challenging process for final customers as they are not aware of the internal organization of the memories they are using, treated as industrial secrets by manufacturers.

To mitigate this issue, a software to reconstruct the internal eMemories organization has been developed. The software takes as input MEUs coming from irradiation test and gives as output a list of possible internal memory organization. The algorithm does so by trying various memory parameters combinations and test if in each combination, all the faults in the MEUs are physically near to each other.

With 5500 sets of input composed of 100 MEUs each (one set per memory organization) the algorithm gives an average of 2 equivalent memory organization, both perfectly capable of simulating the effects of MEUs on the execution of a given program. This thesis will be organized as follows: Chapter 2 provides the reader a background in all the relevant eMemories aspects necessary to understand the presented works and how they contribute to the SOTA. Chapter 3 is the central part of this thesis and contains all the proposed strategies, divided in multiple sections, to advance the SOTA in embedded memories test and diagnosis. In particular:

- Section 3.1 show the in-depth analysis that manufacturers place in the characterization of their eMemories.
- Section 3.2 discuss two novel approaches to compensate Voltage Droops along critical steps in the memory verification flows.
- Section 3.3 presents three highly efficient encoding algorithms for the diagnostic data found during the Characterization process and mass production for statistical purposes.
- Section 3.4 deals with the efficient management of the MBISTs embedded in SoCs to reduce test time and improve fair memory representation.

- Section 3.5 presents a new Machine learning approach for large volumes fault shape classification.
- Section 3.6 explores how memory's MEUs data coming from irradiation tests can be used to reconstruct the internal eMemories organization.

Finally in Chapter 4 there are some final conclusions about the works exposed in this thesis.

Chapter 2

Background and Related Work

This background provides the reader with comprehensive information about embedded memories and several key aspects of their test and diagnosis procedures. The information provided in this chapter gives a solid foundation to understand the proposed strategies shown in Chapter 3.

This chapter starts with the organization of embedded memories and their possible configurations, along with details on their DfT hardware. While the Characterization work shown in Section 3.1 focuses specifically on embedded Flash memories, the methodologies presented in the remaining sections are broadly applicable to all types of embedded memories, as they leverage the regular structures of the bit cells themselves and not on the used technologies. For this reason, the background will primarily address embedded memories in general rather than a specific technology.

In the following section, a description of the traditional testing methodology is presented, together with the current limitations and challenges.

Finally, current SOTA fault classification techniques are introduced. These methods are used to recognize fault shapes found during the tests of the embedded memories. Fault shapes are then useful for test engineers and designers to improve their test flows and strengthen their designs.

2.1 Embedded Memory Architectures and Organizations

2.1.1 Embedded Memories general organization

Embedded memories are organized in regular structures that are easy to replicate and manage [10]. A general overview is depicted in Fig. 2.1. The topmost block comprises several independent memory banks. These are entirely separate structures that can be placed around the die surface. Each bank is then subdivided into several physical sectors (and may contain a Memory Built-in Self Test (MBIST)

to speed up testing operations). Physical sectors are then subdivided into Logical sectors that are divided into Wordlines. Wordlines are composed of multiple words that contain a certain number of Bytes. Bytes represent the minimum granularity accessible by the final user.

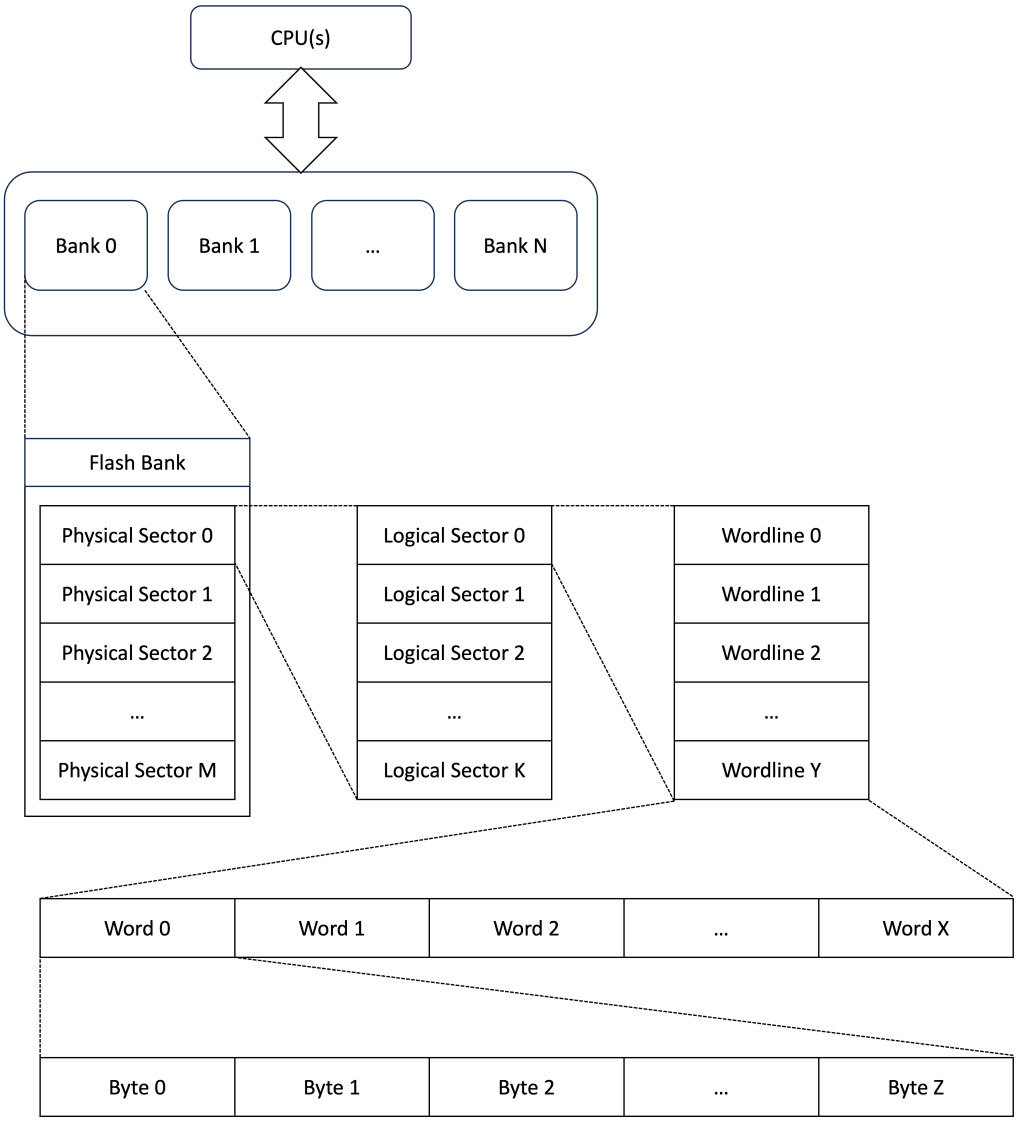


Figure 2.1: A general organization of embedded memories.

2.1.2 Logical vs. Physical addresses

In the programmers' view, the logic memory is a progression of words going from the lowest index to the highest one, as seen in the left part of Fig. 2.2. In this view, the memory is a "black box" in which bits in words are again organized one

after the other from the lowest indexed one to the highest indexed one. Standard memories, however, are organized in a more complex structure to improve their reliability and chances of being repairable with some integrated spare components.

Manufacturers use a set of Memory Design Configurations (MDCs) to modify the organization of their memories. Referring to Table 3.21, some of the MDCs are:

- WORDS: The total number of words in the memory
- MUX: The number of words in a row (also called wordline).
- BITS: The number of bits contained in a single word
- BIT ORDER sequence: Describes how the bits of the words are organized in a wordline.
- Mirroring: This parameter decides if the memory has portions repeated in alternated order. The mirroring comprises either column (bitline) and/or row (wordline) mirroring.
- BIT SCRAMBLING: Bits in wordlines are divided in blocks in which each block i is made of bits with index i^{th}.

Fig. 2.2, 2.3, 2.4, 2.5 and 2.6 show some possible examples of MDCs. In the figures, a single wordline is shown per example, and consequently, only parameters affecting the rows are shown. However, similar configurations can also be applied to the bitlines (column) of the memories.

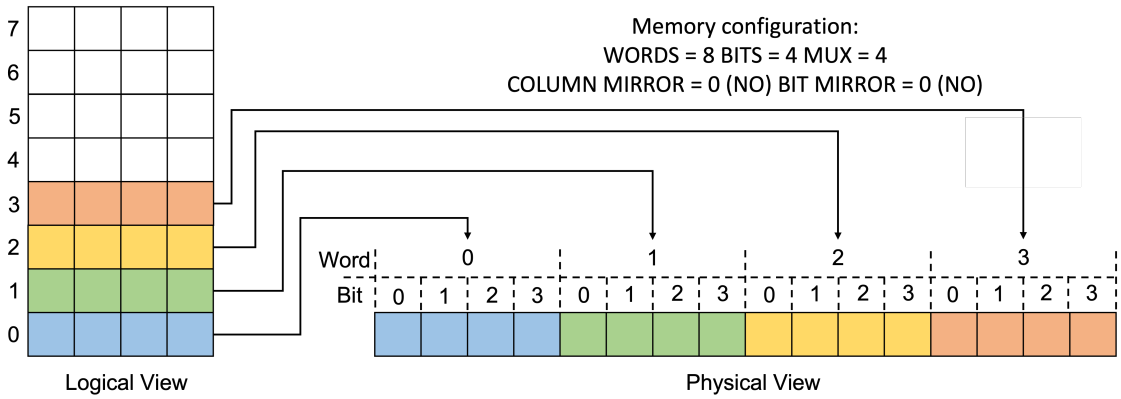


Figure 2.2: An example of an MDC with no bitline or bit mirroring.

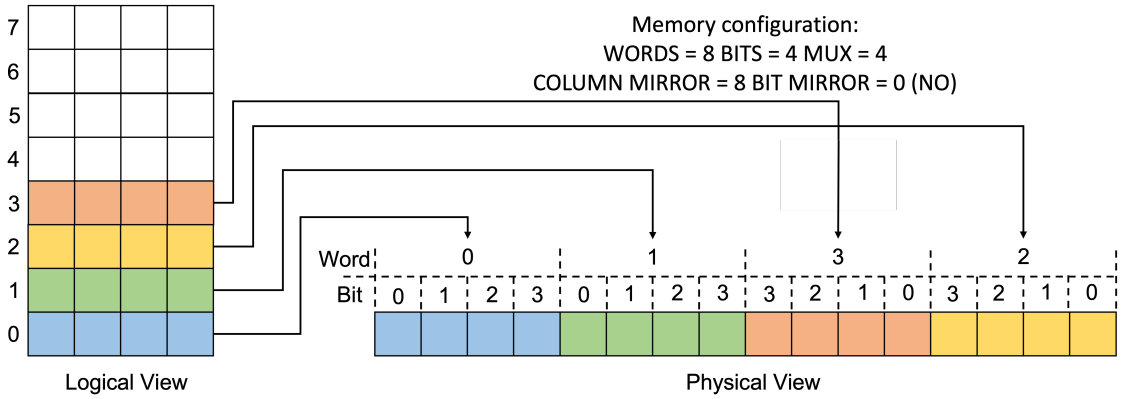


Figure 2.3: An example of an MDC with column mirroring

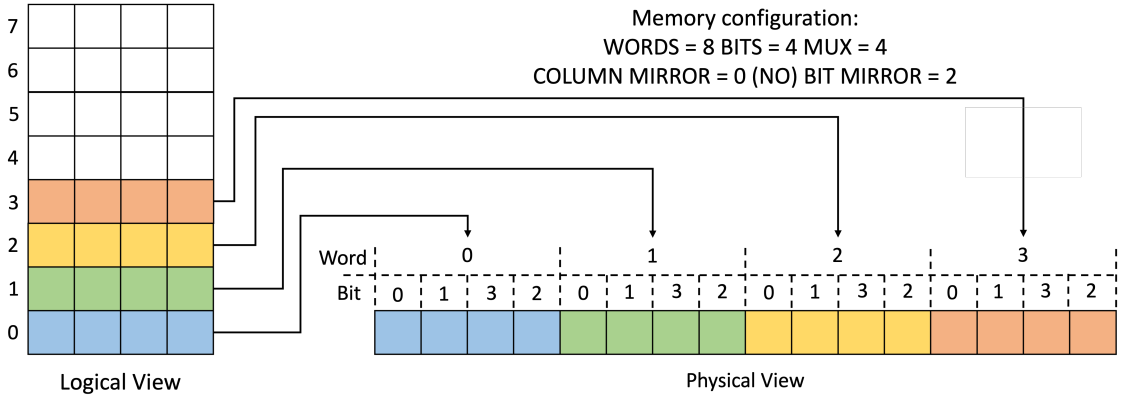


Figure 2.4: An example of an MDC with bit mirroring

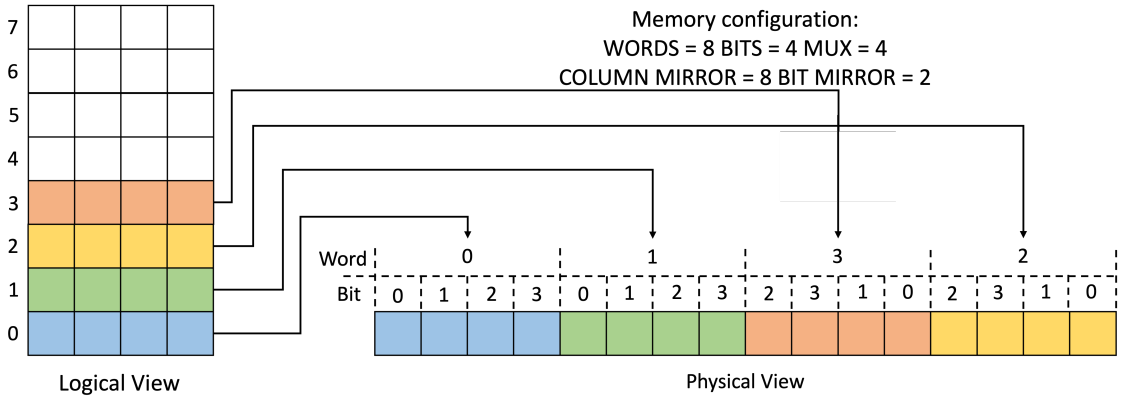


Figure 2.5: An example of an MDC with column and bit mirroring

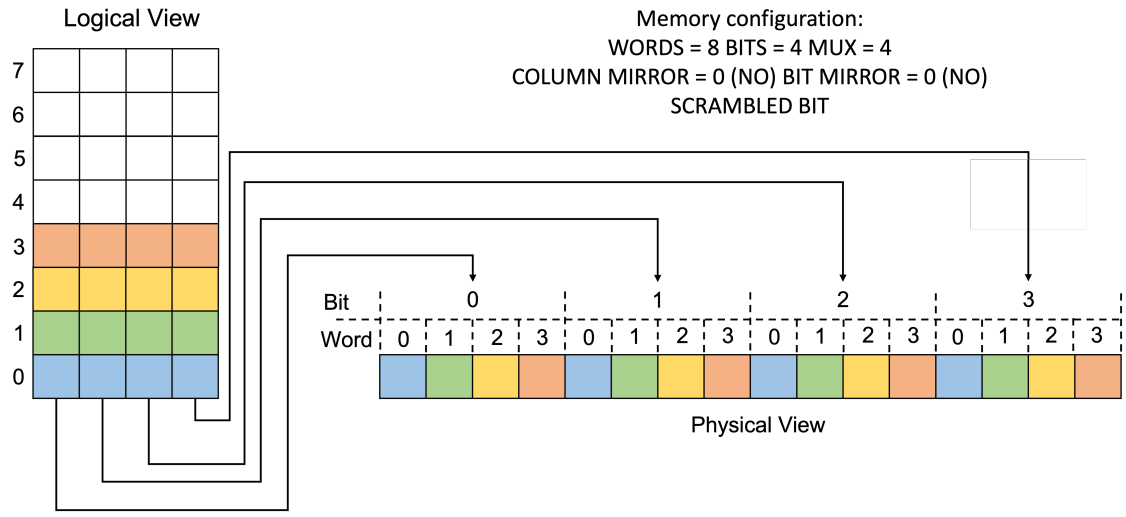


Figure 2.6: An example of an MDC with bit scrambling

2.1.3 Architectures for diagnosis

The most straightforward architecture for embedded memory diagnosis is the one described in Landzberg et al. [2]. This work proposes a methodology based on ATE that can directly access the memory under test by retrieving the fail coordinates as soon as they appear or the collection of coordinates stored on-chip. This method applies no manipulations to the collected data, and the fail constellation can be reconstructed from the whole sets of coordinates.

Differently, Schanstra et al. [15] Chen et al. [16] and Bernardi et al. [17] proposed variations by exploiting additional hardware that integrates the memory test capabilities and supports on-chip bitmap collection. The approach by Schanstra et al. [15] uses a modified MBIST architecture extended to perform shape recognition. The described MBIST recognizes and compresses shapes such as failing bitlines or wordlines. Some faults may be lost during this compression, so this technique does not yield an accurate bitmap representation. Chen et al. [16] propose a compression method to reduce the number of bits needed to reconstruct the failure constellations; this reduction of bits comes at the cost of a lower accuracy of the reconstructed constellations. In Bernardi et al. [17] the authors used the integrated MBIST in combination with the CPU of their device to compact the fault coordinates found during their tests. The MBIST reported the coordinates of each failing bit. The CPU then compacted these addresses, exploiting do not care values by effectively searching cubes of a Karnaugh map. This approach limits the number of communications between ATE and DUT.

2.1.4 MBIST Architectures for Memory Test

Microcontroller eMemory is characterized by highly time-consuming tests. In addition, memory capacity grows quadratically every four years according to Moore's Law and, similarly, testing times also grow.

The introduction of Memory Built-In Self-Test (MBIST) modules led to a test time reduction for which the magnitude depends on the MBIST structure and the CPU's and firmware's ability to access the memory [18, 19, 20]. Consisting of specialized hardware surrounding the memory, MBIST enables at-speed self-testing without the need for expensive and high-frequency Automated Test Equipment (ATE).

Depending on its complexity, an MBIST's logic can apply fixed or programmable test flavors and acts as a slave; thus, it must be configured by the ATE or the CPU, respectively, for production tests and online inspections [3, 4]. The most straightforward configuration of an MBIST is composed of an Address Generator, a Pattern Generator, a Read/Write Controller, and a Data Output Evaluator [21] as shown in Figure 2.7:

- Read/Write Controller is an input to the memory as it decides when its time to write a new test pattern or read it back.
- Address Generator is also an input to the memory as it decides the address for which the test pattern has to be written to or read from.
- Pattern Generator decides the pattern to write at a specific address in the memory. It also provides the reference pattern to the Output Evaluator, which can then compare it with the output of the flash memory.
- Output Evaluator simply compares the output of the memory at a given address with the expected pattern provided by the Pattern Generator. In case of a mismatch, a fault is detected.

MBISTs can access at page granularity and apply patterns and verify the correctness of the read data compared to the expected output. This behavior can be achieved in different ways depending on the MBIST's complexity and can range from simple ATE-controlled MBISTs to more advanced programmable MBISTs able to interact with the CPU through flags or interrupts.

From a higher point of view, MBISTs with programmable options are usually able to implement many modes of operation [22], including:

- Count mode: At the end of the memory test, the MBIST makes the total fail count available to the master.
- Stop-on-fail mode: During the memory test, the MBIST module stops each time a fault is detected and provides the fault's position and type. After a stop event, the master must resume the verification.

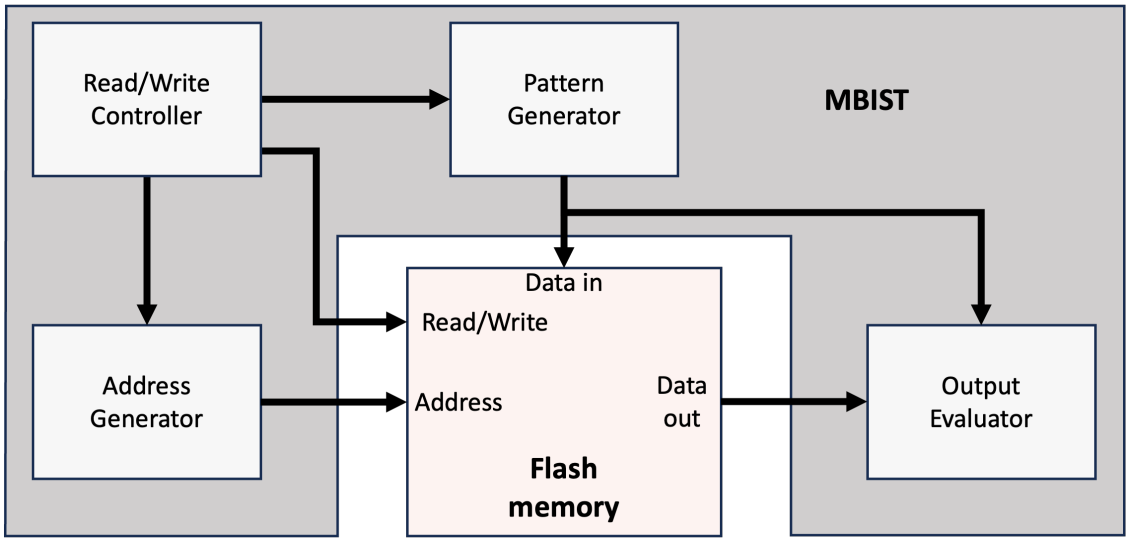


Figure 2.7: Simplified MBIST architecture.

For massive embedded memory composed of multiple banks, the basic structure of an MBIST can be replicated several times. These independent MBIST instances sensibly reduce test time at the cost of a small area overhead. In particular, a distributed MBIST system is better than a centralized one because the test of a fault-free bank is not stopped when a fault occurs in another bank tested by another MBIST instance. The distributed approach gives a significant speed-up during the ramp-up of a technology. An example of a distributed MBIST architecture is illustrated in Figure 2.8.

Nevertheless, coordinating MBIST instances is not a trivial task. Especially for immature technologies, many banks in the same chip may fail simultaneously. Of course, each unit shows its own fail shape, and several MBISTs must be managed concurrently. In a stop-on-fail scenario, MBIST behavior is resumed only after the current fail coordinate is saved. Meanwhile, many MBISTs can notify of failure and wait to be served. This is especially true for devices with thousands or tens of thousands of faults. In these circumstances, efficiently managing the MBISTs can save up to 26% of test time.

2.1.5 Linear Search Method

In the case of multiple independent MBISTs integrated into a system [23], it is necessary to access each one of them individually to check their statuses. This is the case for the system represented in Figure 2.8. The status register is thus the primary communication mechanism used by the MBISTs. After their programming, they only need to write any event (such as “fault found” or “job finished”) to their status register. The CPU can then read the status register of the corresponding

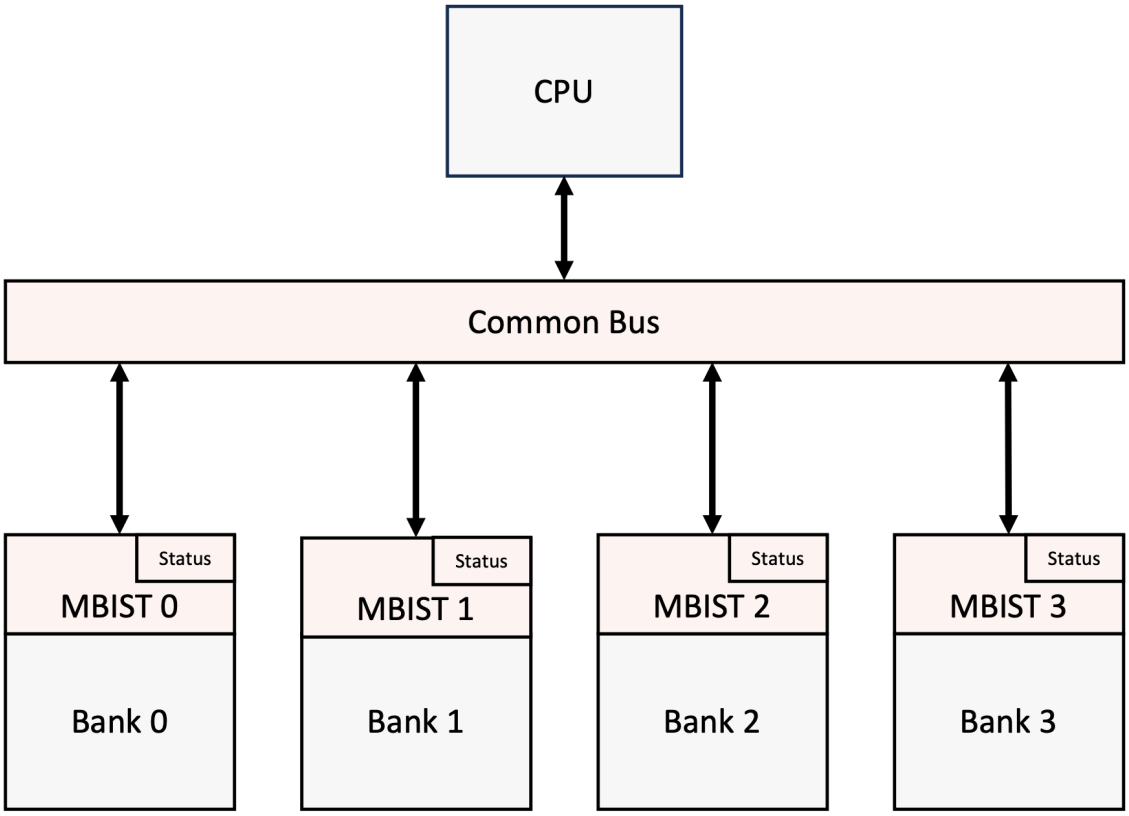


Figure 2.8: Hardware organization of four independent MBISTs.

MBIST. With the independent MBIST configuration set in the stop-on-fail mode, to understand if a fault has been found in the memory bank, the CPU has to continuously check the MBISTs' status register in a linear search fashion.

In the simplest scenario in the so-called "linear search" approach, all MBISTs are checked one-by-one, starting from the one with the lowest index as reported in Algorithm 1.

Algorithm 1: Linear Search for N MBISTs

```
Start all MBISTs;
while Test not finished do
    for each integer i in N do
        if MBIST[i] found a fault then
            Retrieve fault coordinate;
            Restart MBIST[i];
            break;
        else
            Continue;
```

Linear search is the only possible choice for an architecture that requires individual access to the status register of its MBISTs. A perfect example of this kind of architecture is presented in [5], where the MBISTs are connected to the central controller bus through a multiplexer that selects the desired MBIST on a one-by-one basis. Thus, no parallelization during the MBISTs' status check is possible using the architecture proposed in [5].

To visualize how the linear search works and its intrinsic issues, Figures 2.9 and 2.10 show how this access mechanism works in a possible test scenario with many fails distributed along multiple memory banks. This scenario is an example of what automotive SoC manufacturers may find during the characterization of their devices. Referring to Figure 2.9 in particular:

1. Banks 0 and 2 represent randomly distributed faults with various shapes.
2. Bank 1 has a faulty bit cell at the junction of the "cross" of failing bit cells. That faulty bit cell appears to work correctly but negatively affects nearby cells.
3. Failures in Bank 3 represent a failing sense amplifier.

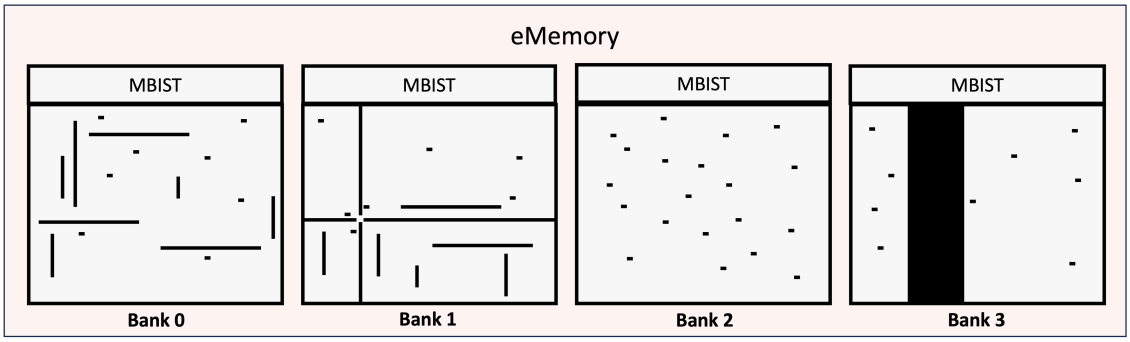


Figure 2.9: Realistic fault scenario in 4-bank memory.

The diagram in Figure 2.10 shows a possible order in which the banks (or better, their MBISTs) requiring CPU attention are served:

- T_0. At the start, each MBIST is in the running phase and is testing the memory in search of faults.
- T_1. The MBIST of Bank 0 is the first to find a fault; it stops the bank test and waits for the CPU read. The other MBISTs are still running and are searching for faults.
- T_2. The CPU accesses MBIST 0; MBIST 3 stops and waits while the other MBISTs are still running.

- T_3. The CPU resumes MBIST 0. All the other MBISTs are stopped and are waiting for CPU intervention.
- T_4. The CPU starts analyzing MBIST 1. MBISTs 2 and 3 are stopped and are waiting for the CPU read, while MBIST 0 is again running and is searching for faults.
- T_5. MBIST 0 stops and is waiting for the CPU, MBIST 1 is running. MBISTs 2 and 3 are still waiting for CPU intervention.
- T_6. The CPU starts analyzing MBIST 0. All other MBISTs are stopped and are waiting for CPU attention (MBIST 3 has been waiting since T_3).

After T_6, if MBIST 0 and MBIST 1 continue to find faults, they will always be served first. With this intrinsic priority, MBISTs 2 and 3 will be served last. This unfair serving of the banks is especially problematic in cases for which there are time or diagnostic memory limitations (i.e., if there is not enough space to store all fault information, faults appearing in Banks 2 and 3 will not be logged).

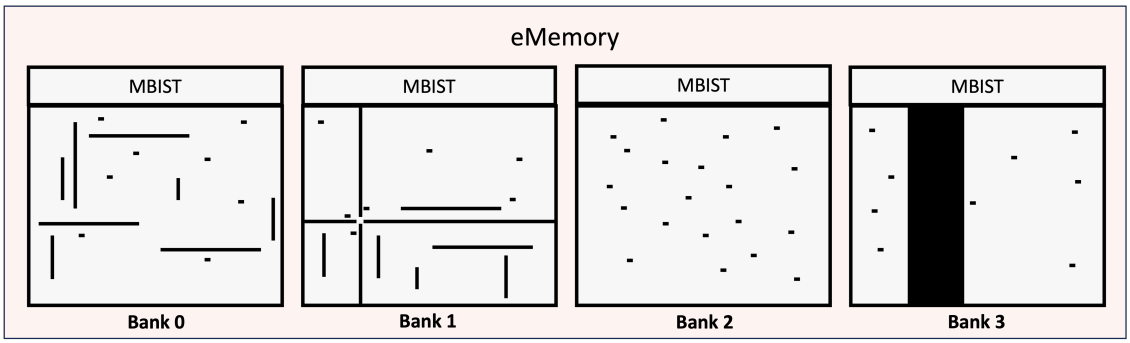


Figure 2.10: Timing diagram of the linear search applied to the MBISTs of a fault-dense 4-bank memory.

2.2 Traditional Memory Testing Methods and their Limitations

2.2.1 Measurements setup

During their tests, manufacturers adopt a complete array of measurement devices. Having a stable and reliable setup is essential as it is fundamental to guarantee the repeatability of the measurements. Measuring instruments include:

- Oscilloscope with multiple channels to be used, for example, for current and voltage measurements. The oscilloscope can also be connected to the DUT's

General Purpose Input/Output pins (GPIOs). These GPIOs can then be used as triggers to understand the starting and ending point of the test and the exact time a failure has occurred due to a voltage droop.

- Current probe, an instrument designed to be clamped onto insulated conductors and used to precisely measure the current. Current probes are more precise than shunting the input terminals with a low-value resistor and reading the voltage droop over its terminals.

2.2.2 Test Characterization

SoCs can work at different combinations of frequencies and supply voltages. A combination of high frequency and supply voltage is chosen for high performance, increasing the power consumption and, consequently, the device's temperature. On the opposite, when power consumption has to be minimized, the supply voltage and the working frequency are lowered. During validation and characterization, one of the main tests for Automotive SoCs is called voltage sweep, a stress test aimed at evaluating the behavior of the DUT at the variation of the supply voltage. The absolute maximum and minimum voltage values are determined by observing the correctness of the results of a given set of operations ranging from memory read-/write to logic computations. These tests are fundamental to characterize a device's performance and power consumption, especially for the lower voltage values. Manufacturers test their devices in harsher conditions than the one recommended for the final users to retain a certain safety margin. During this characterization phase, a minimum voltage called V_{min} is chosen.

V_{min} is chosen considering a Guard band that allows the device to work even if a relatively small voltage droop is present and is equal to $V_{fail} + V_{Guard\ band}$ (Fig. 2.11).

2.2.3 Voltage droops and decoupling capacitors

All SoCs have minimum and maximum operating voltage specifications. If the voltage is lowered under a minimum threshold, the device no longer work reliably.

To keep the voltage stable, it is a good habit to insert capacitors between the supply voltage and the ground. These capacitors are able to compensate a certain degree of voltage variations between their terminals as graphically shown in Fig. 2.12. These so called decoupling capacitors are able, when the input voltage droops or increases, to respectively provide or absorb the excess energy that is trying to flow through to the SoC. If the voltage droop is too quick and extended, the capacitors are not able to compensate, and the device goes in an undervolt condition that triggers an alarm, consequently failing the test.

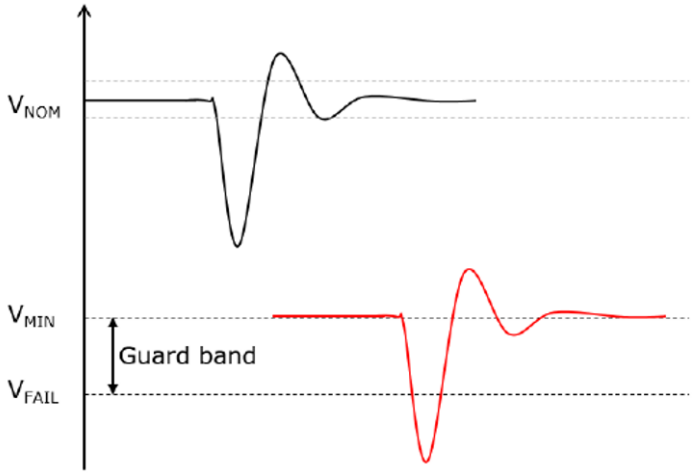


Figure 2.11: Effect of voltage droops at different supply voltage

The droops are caused by a high dynamic power consumption that depends on the executed instructions. When this happens, the device requires a large amount of current for a brief period from the power supply that is not fast enough to prevent a voltage droop.

There is a strict correlation between a rapid requirement of current and voltage droops. Current spikes are usually caused by strong activity inside the devices, like activating more than one CPU in parallel or performing certain energy intensive steps. The correlated current spikes cause a discharge of the decoupling capacitors that cause a voltage droop according to the formula $V = Q/C$. In the long run the power supply will compensate this droops, but if their amplitude is high enough the device will show misbehavior.

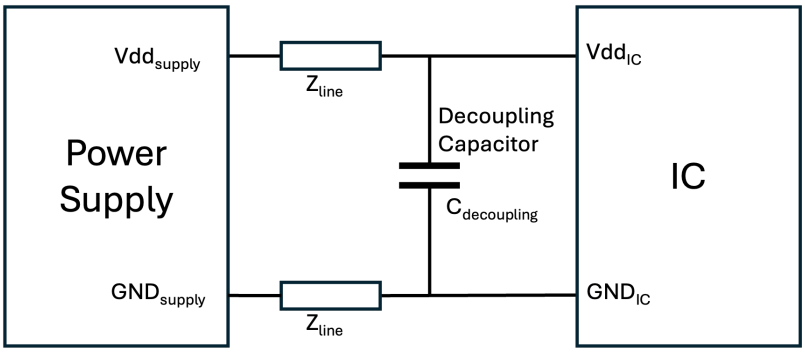


Figure 2.12: Power supply setup

Voltage droops are considered critical when testing at the minimum voltage because when a voltage droop happens, the test is no longer performed at minimum voltage but at a lower one. In this case the voltage can reach V_{fail} causing fails.

2.2.4 Characterization and Memory Current Margin Test

Memory units may present many faults in the early steps of technology development. Characterization processes are used to perform studies about statistical topological distribution that may be present due to non-optimal architectural design or specific process variation or different working parameters (temperature, frequency and so on) [1]. Many verification steps are performed to find fault distribution when changing the Sense Amplifier reference current and observing the corresponding bit misbehavior. This process can highlight recurring defects in the memory behavior, such as non-uniform topological faults distributions, that are process and technology-dependent.

The resulting diagnostic information can be seen as coordinates that describe the physical location of the faults present for the selected margin.

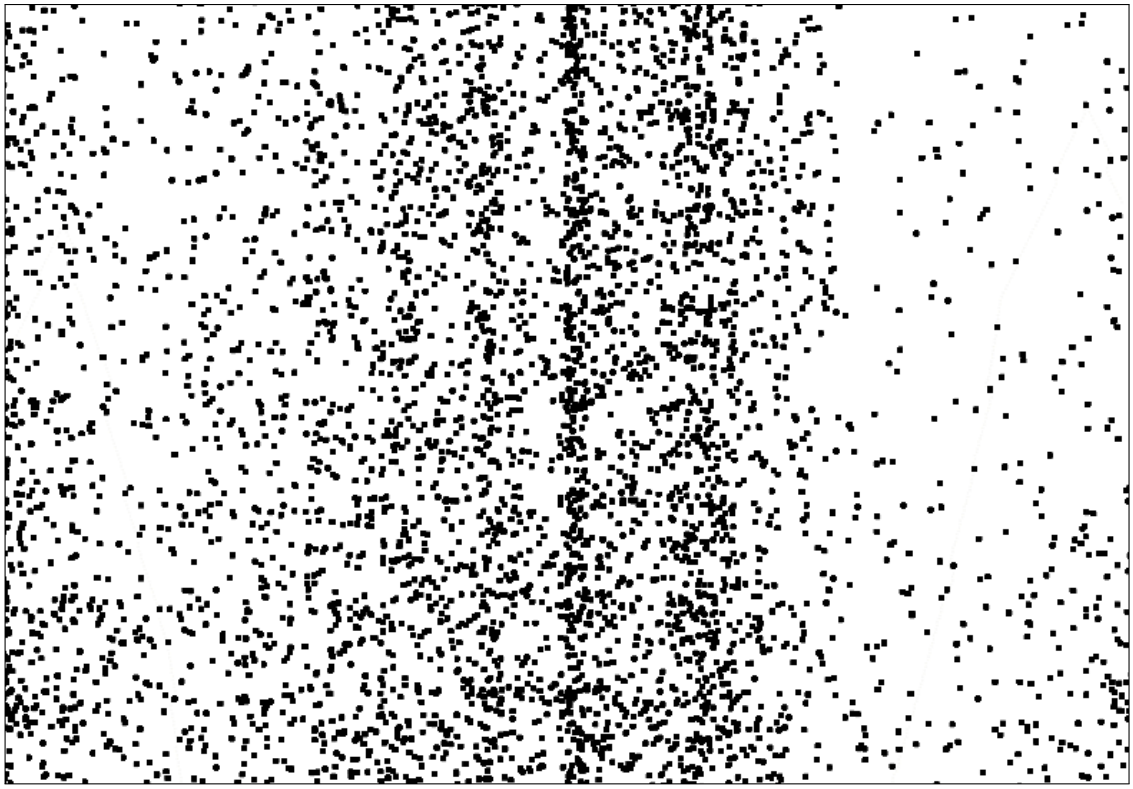


Figure 2.13: Standard bit fail map generated shifting the reference current of the Sense Amplifiers

2.2.5 Fault topological distribution

Technology experts are interested in topological failings patterns with respect to their exact positions. They benefit from a compressed overview of the failings

in the memory that shows the fault density more than their precise locations. The benefit is especially evident for fault-dense DUTs, where the number of faults limits the ability to represent a large portion of the memory. Also, the exact coordinates of every fault are not needed for this type of characterization.

A topological failings pattern corresponds to an increased probability of faults in a specific area of the physical memory. Multiple factors can interfere with the correct reading, altering the programming state of the cells and then generating a topological failings pattern. Fig. 2.13 represents the failing information using a standard Bitmap in which each black dot corresponds to a failure. Failures are focused on the center, highlighting a topological failing pattern.

2.2.6 Test flow

Since requirements for automotive fields are particularly strict, eMemories have to be reliable across various environmental and electrical conditions. The reliability of the embedded memories has to be assessed in conditions such as:

- Temperature: standard operating temperatures for embedded memories range from -55°C to 125°C. During characterization and mass production, DUTs are actually tested in temperatures slightly above and slightly below the specified ones to maintain a certain safety margin.
- Voltage: also for supply Voltage there is a specified range of acceptable values. To assess the correct behavior of the devices along the entire range, manufacturers test values slightly below and slightly above the specified values.
- Timing: Memory readings or writings follow a set of precisely timed operations involving multiple actors such as the sense amplifiers, the row and column selectors, and, generally, signals propagation. The correct timing behavior has to be deeply tested during characterization since is one of the key aspect of every memory read or write operations. Fig. **??** (from [1]) illustrates how supply voltage and read access time affect the fault distribution in an Infineon™ Automotive SoC embedded memory.
- Sense Amplifier reference: a circuit named Sense Amplifier (SA) is used to decide whether the content of a bit cell is a 0 or a 1. The SA compares the current of the read memory cell against a reference current. If the cell's current is higher than the reference, the bit cells is read as 0, 1 otherwise. The manufacturer sets the reference current depending on the technology, and its correct calibration is fundamental to obtaining a valid read from the memory.

To test all these aspects, a comprehensive test flow is developed. The test flow is a collection of multiple test steps performed in different configurations. Apart

from testing the DUT's behavior at different environmental conditions, these test flows also search for faults such as the classic stuck-at and slow-to-raise faults.

For these reasons, automotive SoC manufacturers have to test the embedded memories of their devices deeply. Multiple sensitization and detection steps are performed in different conditions to test the memories fully. The sensitization consists of:

- Active: program, erase, disturb, and stress operations.
- Passive: Retention bake and accelerated read disturb.

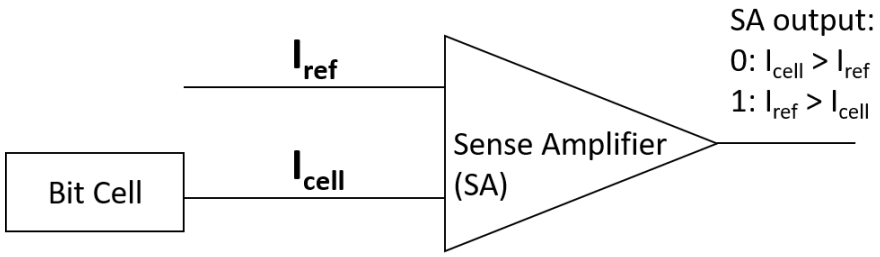


Figure 2.14: The sense amplifier decides if the content of a bit cell is a 0 or a 1

As described in Figure 2.15, these sensitization-detection steps are performed both in the Front End, at the wafer level, and in the Back End after packaging.

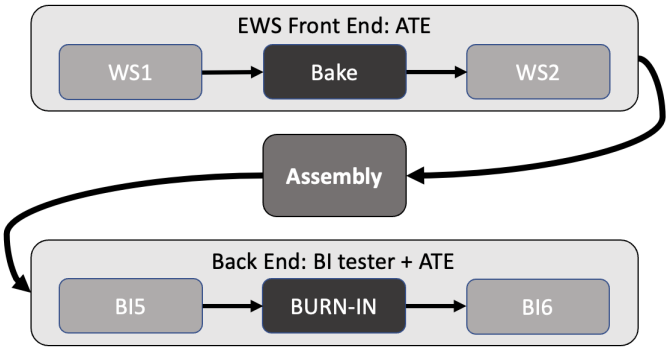


Figure 2.15: eFlash test Flow from Wafer level to Package level

The main goal of these tests is to discard defective devices as soon as possible, to optimize the process costs.

The first step is the Wafer Sort (WS), in which the manufacturers skim the first defective units. This is followed by the data retention Bake and by another WS to help assess the data storage capacity of the memories under test.

Devices that survived the previous steps are then assembled (packaged) and tested again in the Back End (BE) after a Burn-in that stresses the memories with a series of memory operations. Fig. 2.15 graphically explains this test flow.

2.2.7 Diagnostic data encoding methods

The test flows in the preceding paragraph produce a large volume of diagnostic data. This data must be managed effectively by the manufacturers who seek to both cut test time and boost their yield (and consequently reduce test cost).

To handle this data there are multiple methods found in the literature that span from list-based methods [2] to more advanced shape recognition methods [24] to lossy compression methods[25].

List-based methods

List-based method such as the one presented in [2] are the easiest ones both conceptually and from an algorithmic point of view. Each fault found is represented in a list together with its coordinates. The advantage of this method is in the immediate processing of the data, which just requires storing the fault location without any processing. The disadvantage lies in the memory requirements to store all the fault location one by one. Faults that involve an entire bitline or wordline or even a physical sector can take a huge amount of space to be represented. Fig. 2.16 shows an example of a failing bitline and some sparse faults represented with Landzberg's method.

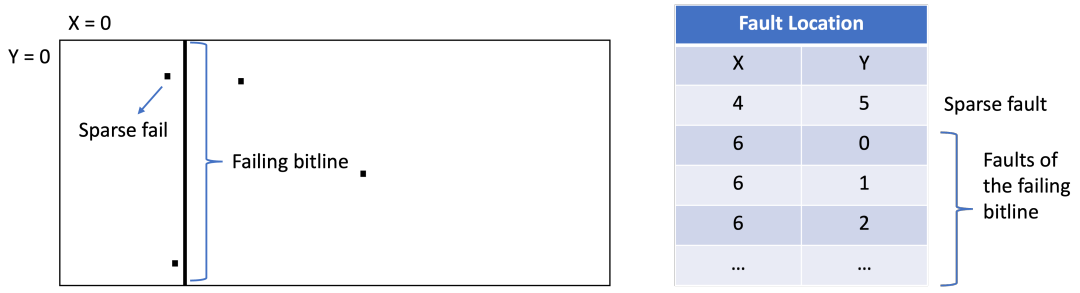


Figure 2.16: In the Landzberg's representation each fault location is reported individually

2.3 Existing Fault Classification Techniques

Industrial approaches to large-scale bitmap analysis of memory errors are typically based on heuristic methods, in which carefully crafted algorithms are developed by experienced engineers. These algorithms are good at processing large volumes of faulty logs to detect and classify "known" faulty shapes emerging from tests of memories. Nevertheless, they suffer from (i) poor performance as they rely on a series of nested loops (i.e., very long execution time), (ii) difficulty in detecting new classes of defective forms (i.e., heuristic algorithms can detect the faulty

shapes they have in their database). As an example, LI-C. Wang in [26] discusses the engineering tasks in the production yield application context. Every wafer goes through a sequence of tools (manufacturing equipments), but there could be more than hundreds of tools involved. It underlines the necessity of both automating the workflow and supporting the data analysis with machine learning algorithms. It is worth to underline that this is task-dependent: many applications in design and test do not have a "big-data" problem [27]. However, the problem we are addressing do suffer from such issue.

2.3.1 CNN-Based Classification of Memory Failure Bitmaps

As a matter of fact, in recent decades, to facilitate and aid this diagnosis process, traditional industrial heuristic algorithms have been powered with machine learning approaches. The intent is manyfold: to predict bit errors, to increase the yield of a memory device, to automate and support the test chain, and so on. ML-based approaches have been widely explored for accurately predicting endurance levels or reliability issues in Flash memories [28]. In [29], a polynomial evaluation model for predicting NAND Flash bit errors is proposed. A further prediction scheme based on the SVM algorithm is presented in [30] to predict endurance levels of Flash memories.

When it comes to diagnosis, the majority of research works focuses on the detection of systematic defects on production *wafers*. This is carried out by means of statistical approaches (e.g., [31]), machine learning and more advanced deep learning algorithms (e.g., [32, 33]). The authors in [31] adopt a layout-aware diagnosis based on statistical criteria to wafers' diagnosis to provide more resolution and eliminate ambiguities. Surfing the high wave of AI technology, Nero et al. in [32] proposes a technique for *wafer diagnosis* based on Generative Adversarial Network (GAN). The adopted GAN leverages on two CNNs for building the generator (G) and discriminator (D) neural networks, accounting for a total of 34,067,601 trainable parameters. The main limitation of the work resides in the dependence on the classes of images used in the training (it may demand a very large training set, leading to a long training time). The authors also discuss the potential robustness issues linked to adversarial samples. In [9], the authors explore the potentiality of Artificial Neural Networks (ANNs) to classify shapes over a quite large memory matrix. To survive such a memory size, the method does not feed the ANN with an image but is based on the extraction of several features from the failure bitmap under evaluation. A pre-processing part based on clustering the faulty portions of the memory. For each identified region, 4 features are extracted, including the size and density of the failing region. Each region is separately classified according to *known* shapes. The process is fast and quite accurate, despite "blob" failures introduced by the authors as "unknown" shapes that pull down the accuracy of the classification. Anyway, it is so far the only paper looking for a concrete way to ease

the inspection for novel failure shapes, not yet classified.

Among the different AI applications, image processing has gained significant popularity in recent years in different fields, from medical imagery, to agriculture industry. Specifically, CNN-based image categorization techniques have progressed as well quickly [34]. As a matter of fact, CNNs excel at learning the structural representations of images and accurately predicting their contents [35], by leveraging the core operation: i.e., the convolution. Making use of filters (also known as weights) to identify and examine particular patterns in the image, they are highly effective for extracting hierarchical features from images.

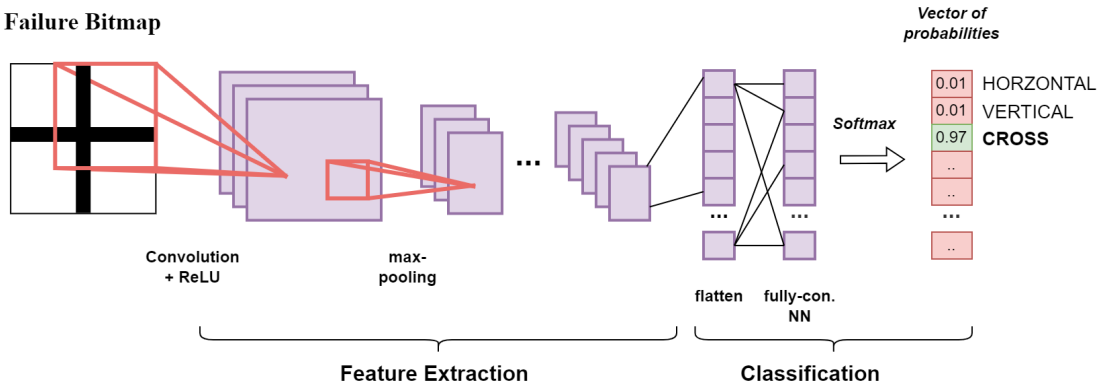


Figure 2.17: Exploiting CNNs to classify memory failing bitmaps.

Fig. 2.17 depicts the process of classifying the shape of a failing memory bitmap, in our specific context. The first block, i.e., feature extraction, typically intermixes convolutional, ReLU, and pooling layers to extract features from the analyzed image (e.g., shapes) using pre-trained filters (i.e., weights). Then, a classification block (consisting of fully connected layers, followed by a final softmax function) transforms the final values into a vector of probabilities; the probability assigned to every label is also known as the confidence level. The higher the confidence level, the higher the probability of a correct selection of the class. Fig. 2.17 illustrates an example of correct prediction: the picture represents a multi-adjacent bitline, which corresponds to the CNN's highest score after classification; the confidence level is 97%, e.g., 0.97 probability of having a correct prediction.

Chapter 3

Proposed strategies and experimental results

This chapter presents all the research performed to advance the state-of-the-art in the embedded memory testing field, together with their experimental results.

Embedded memory testing is a complex field, especially in safety-critical environments such as the automotive one where the reliability of the electronic components is paramount, and their deep assessment is mandated by standards such as the ISO26262. To reach the goal of advancing the SOTA in the memory test and diagnosis, it is necessary to tackle this challenge from several different perspectives. For this reason, this chapter is comprised of works in multiple key areas of the embedded memories life cycles:

- Section 3.1 concentrates on the Characterization of the embedded memories a process that precede the mass production of the devices. Before starting mass production, it is paramount to determine the exact characteristics of the memories produced and to compare them with the results obtained in simulation environments. During this characterization process, several key parameters are set, including the reference current for the sense amplifiers. The memories are characterized across multiple supply voltage and temperature ranges to evaluate the dependence of key parameters.

- Section 3.2: Part of the characterization is performing tests of embedded memories at a low supply voltage called V_{min}. This section tackles the issue of voltage droops along with the testing of embedded memories at this low voltage level. In this kind of test, the manufacturers change the supply of their SoCs to a limit close to the minimum operating voltage of the devices and consequently stress-test their embedded memories. While performing these tests, the activation of the DfT hardware and some energy-intensive preparation steps lead to voltage droops. This phenomenon may lower the voltage below a critical level, and so, in turn, cause false fails unrelated to the

memory under test. To solve this issue, two circuits have been proposed to temporarily raise the supply voltage when a critical voltage drop is expected and then to lower it again after this critical step is passed. Compared to previous SOTA solutions, these two circuits are simple, small, and easy to integrate and work very efficiently in test environments with predetermined test flows.

- Section 3.3: After stabilizing the tests to solve the false fail issue, the collection of the data generated during the test flow is tackled. During a test flow, immense amounts of diagnostic data are generated. Efficiently managing the data is paramount to limit the communications between the SoCs under test and the external tester, resulting in faster test times and lower costs for manufacturers. In this section, a series of three different algorithms are presented: two of them losslessly encode the faults found in the memory based on their shape and (the most efficient one) on their repetitions across several tests, while the other is lossy and density-oriented and is incredibly efficient to show fault distribution along the devices in an incredibly efficient way. Each algorithm sensibly improves the previous SOTA solutions.

- Section 3.4: While the previous section concentrated on the diagnostic data encoding, the work in this section is related to the diagnostic data collection and how to efficiently manage the DfT hardware to save even more test time and have fairer representation of the memory conditions. This result is achieved by an innovative hardware-software scheme that enables the parallel polling of several Memory Built-In-Self-Tests (MBISTs) and a periodical priority-shift mechanism that guarantees the periodic serving of all the MBISTs needing the SoC's attention without the intrinsic priority shown in previous SOTA approaches. T experimental results prove a sensible reduction in test time together with a fairer representation of the memory.

- Section 3.5: Once diagnostic data coming from embedded memories are efficiently collected and encoded, it is time to use them to categorize the kind of failures encountered during the embedded memories tests. Test engineers and designers will then use this information to improve the robustness of their designs or to concentrate the test efforts in the most relevant areas. This section proposes a novel CNN network based on the ResNet18 architecture to categorize fault shapes into 8 different categories. The network is also able to recognize previously unseen fault shapes to help engineers quickly identify new fault mechanisms. Compared to previous SOTA solutions, the proposed method performs comparably to the best algorithmic solution while being one order of magnitude faster (important for huge volumes of data) and incredibly flexible, allowing for the easy addition of new fault shapes.

- Section 3.6: All the previous sections tackled the issues that manufacturers encounter during characterization and general tests of the memories embedded in their SoCs. This section tackles the test issues of the final programmers of the boards and the integrators that need to use these SoCs in their products (for example cars' ECU manufacturers). After manufacturers made huge efforts to ensure the reliability of their devices, a difficult-to-solve reliability issue persists: cosmic radiation. During the operational life of the devices, it is expected that highly charged particles will interact with the devices' transistors, causing sudden faults inside the devices. When the highly charged particles interact with the embedded memories, in particular, they risk altering several adjacent bits at the same time due to the closeness of the cells and the regular shape of the memory structures. This phenomenon takes the name of Multiple Bit Upsets (MEUs). Developers cannot easily simulate the effects of MEUs as they would need to know the internal organization of the memories themselves, a detail that is rarely shared by SoC manufacturers. The only reliable way to perform this simulation is to bring the devices to dedicated radiation laboratories, a process that is not only slow but also very costly and should be repeated with each new version of the code or new application. To solve this issue, this section will introduce an algorithm to reconstruct the Memory Design Configuration (MDC) from a limited amount of information from a single and quick radiation experiment, and so from a list of MEUs and their logical addresses in the memory. Using the reconstructed MDC information, developers can efficiently simulate the effects of MEUs on their applications. Experimental results show the validity of our approach using simulated memories with different MDCs: the algorithm always returns the correct MDC together with some "equivalent" ones (the number of equivalent MDCs decreases with the increase of MEU data provided to the algorithm).

All the work described from Section 3.1 to Section 3.6 shows experimental results coming from Infineon Aurix devices and stems from a multi-year collaboration with Infineon Technologies that advanced the memory testing SOTA in several key aspects, while the validity and efficiency of the work in Section 3.6 is proved through the simulation of thousands of memories with different MDCs.

3.1 Defect-Oriented Testing and Memory Characterization

This section introduces the characterization steps for the flash memories embedded in automotive SoCs. During this phase, the various specifications of the memories are assessed to verify their compatibility with the values simulated during the design phase. Characterization is a time-intensive process and is performed

on a limited sample of dies, typically involving new technology nodes or memory technologies.

The first section focuses on the Erased Cell Current Distribution, a metric used to calibrate the reference current of the sense amplifiers in the memories. In this step, all memory cells are erased, and the current they generate is measured by incrementally shifting the reference current of the sense amplifiers in small steps.

The second section addresses the characterization of the Read Access Time, a metric used to evaluate the speed of the memories. Factors such as internal wiring and sense amplifier delays impose a lower threshold on the operational speed of a memory. Faster read speeds can lead to errors in bit decoding, where bits are erroneously interpreted as 0s even when they contain 1s.

All the results shown in this section have been collected on Infineon Tricore devices and the work resulted in a publication at IEEE IOLTS 2022[1].

3.1.1 In-depth Analysis of Erased Cell Current Distribution

In a Flash memory the value of each bit cell is stored in a floating-gate transistor, which can hold a charge to represent data. These transistors are organized in row (wordlines) and column (bitlines). During a read operation, the memory controller activates the wordline and bitline corresponding to the target cell. A sense amplifier then compares the current generated by the selected cell with a reference current to determine the stored value. If the current generated by the selected bit cell is higher than the reference one, then the bit is decoded as a 0. Vice-versa, if this current is lower, it is decoded as a 1. This mechanism is explained in Fig. 3.1. This current measure may vary from cell to cell. Therefore, analyzing the overall cell matrix values and distribution is essential.

Identifying the most appropriate reference current to use during volume testing is one of the characterization steps performed on the eFlash. The test of the current observed from each memory cell after an erase or a program operation provides valuable information to set the reference current limit for an erased or programmed cell.

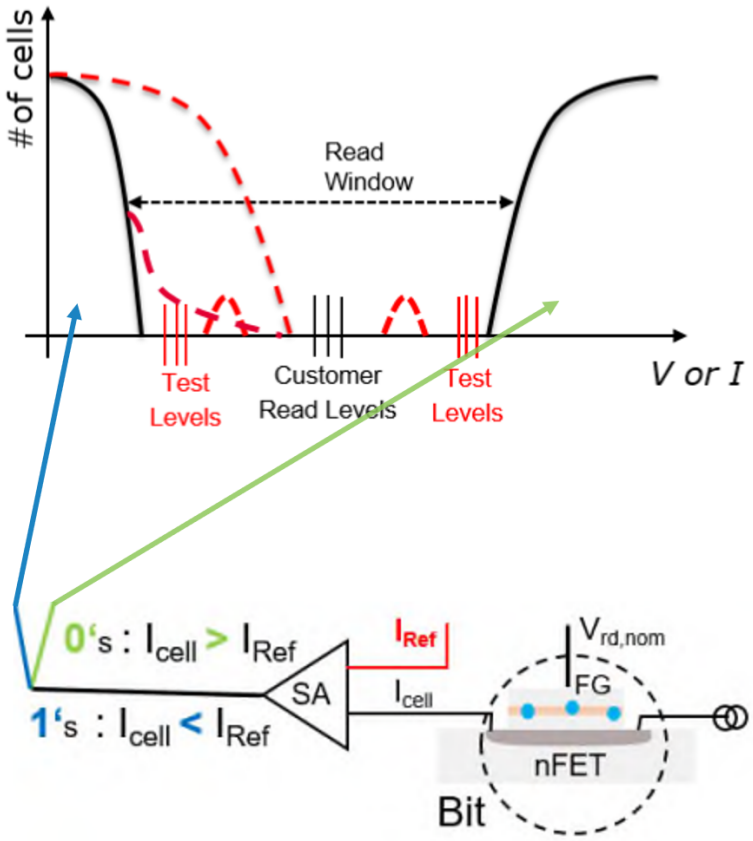


Figure 3.1: A sense amplifier, part of the bit cell decoding mechanism

The result of the current tests are collected through stacked bitmaps classically collected through bitmapping techniques [2] or with more advanced shape recognition techniques [24]. The following Fig. 3.2 shows an example of a stacked bitmap, showing the current distribution in an erased bank. In the figure, the intensity of the violet color corresponds to the magnitude of the bit cell current. The more violet, the higher the current. A bit cell is recognized as a logical '0' if its current exceeds the reference current of the sense amplifier. Therefore, a higher bit cell current indicates a stronger bit cell.

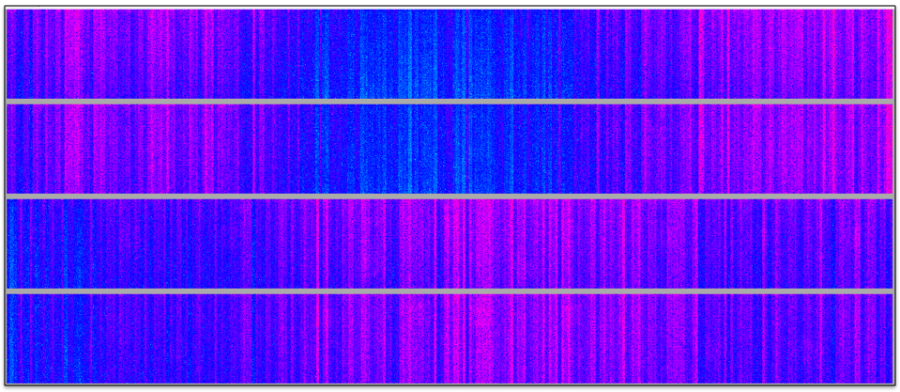


Figure 3.2: Stacked bitmap showing the current distribution of erased cells in a bank. The more violet the color, the higher the current of the cell

As it can be seen in the figure 3.2, the current distribution is strongly position-dependent. With "weaker" cells concentrated in specific areas such as the middle part of the top half of Fig. 3.2. Manufacturers can then use the information coming from these stacked bitmaps to set the lower limit for their sense amplifier reference currents. Erased cells that exhibit a current lower than this limit will be marked as defective.

3.1.2 Characterization of Read Access Time

Another crucial aspect for manufacturers is the read access time for the eFlash memories. The faster it is, the higher the performance of the SoC they are part of. The lower limit for the access time is established by multiple factors such as wires and sense amplifiers delays. However, also these parameters show some process variations together with temperature and voltage dependence and need to be assessed during the characterization of the devices.

To perform this characterization, the manufacturers can act on different aspects of their memory read, such as:

- the supply voltage of the eFlash modules;
- the temperature at which the characterization is performed;
- the system frequency of the SoC they are part of.

The following example in Fig. 3.3 shows the effect of changing the system frequency and the reference voltage on the eFlash behavior when operating at room temperature. Along with these eFlash tests, performed by stepping through growing voltage and frequency values, the interesting parameter is the point at which faults start to appear.

In Fig. 3.3 the horizontal ax represents the frequency at which the SoC is working: faster frequency means less time for the memories to correctly return their read result. The vertical ax represents the percentage of failings over the total amount of tested bits. The multiple semi-circular colored shapes represents fault appearance at different voltage levels. The green curve represents the access time of the CPU, how fast the CPU read the content of the memory after requesting a read operation (decreasing with the increase of the system frequency). The red constant line represents the lower limit for the CPU access time to read meaningful data as it is the actual time at which the memory starts to output the read results, but fault starts to appear sooner due to other delays. As emerging from 3.3, the voltage has a clear effect on the performance of the eFlash. The lower the voltage, the sooner the faults start to appear. On the other hand, by focusing on the frequency, the figure shows that at some critical points, the eFlash has not enough time to decode the bits correctly, and failures start to appear.

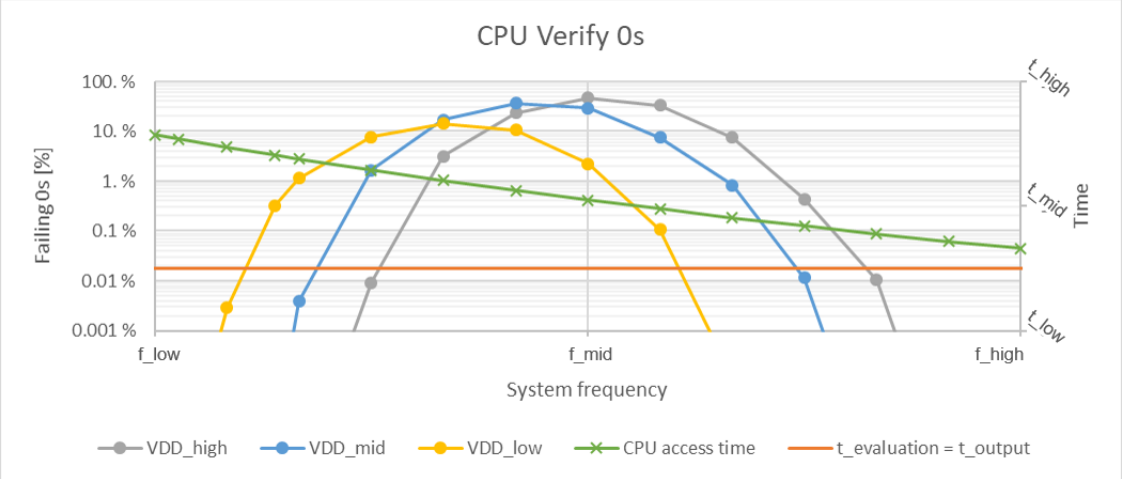


Figure 3.3: Example of eFlash characterization tests performed at room temperature

Most manufacturers also include additional hardware components called Memory Built-in-Self-Test (MBIST) in their eFlash to speed up the testing procedures. Nevertheless, also the performance of these devices has to be characterized. Fig. 3.4 shows the results of these tests in case of a verify 0s on a memory composed of all erased cells. Similarly, Fig. 3.5 shows a verify 1s over a memory composed of all programmed cells.

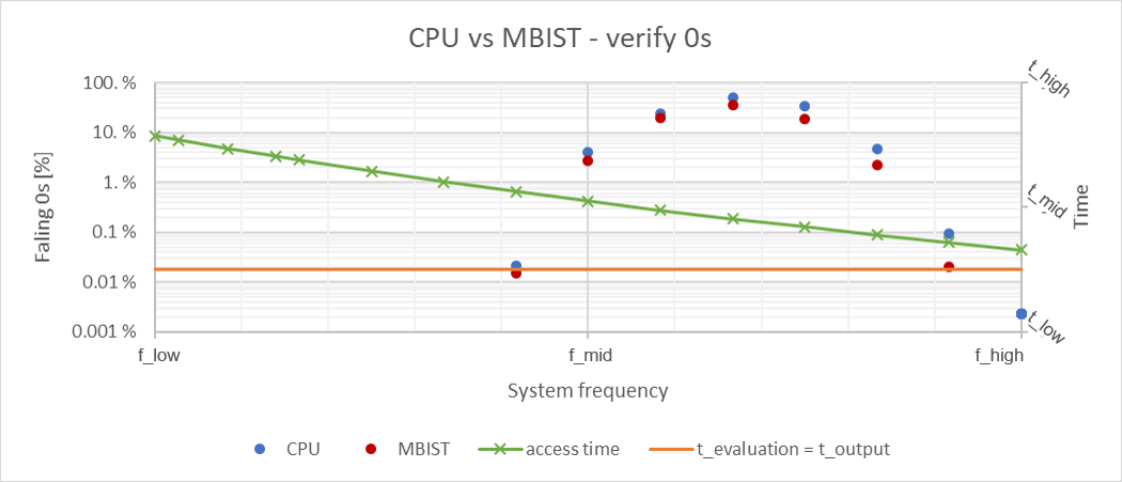


Figure 3.4: Test to check the effectiveness of the integrated MBIST against the CPU in case of a verify 0s

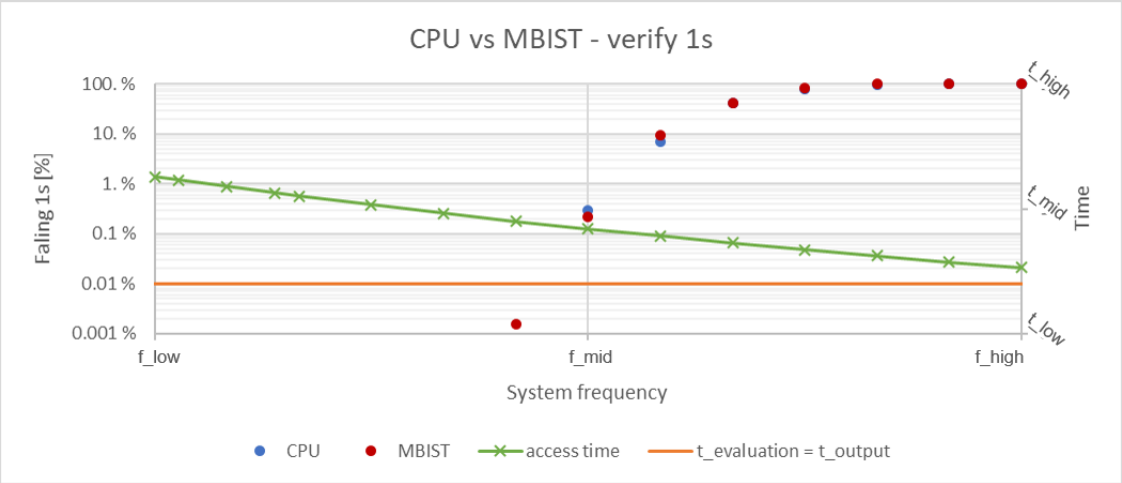


Figure 3.5: Test to check the effectiveness of the integrated MBIST against the CPU in case of a verify 1s

The previous two figures show that the CPU and MBIST results are almost identical. The comparison between CPU and MBIST shows that the specialized hardware can be reliably operated in the testing environment, with all the advantages in test time it brings.

3.2 Voltage Droop Mitigation Techniques

Voltage droop is a critical challenge in modern integrated circuit testing, particularly when operating at the minimum supply voltage (V_{min}). This phenomenon, caused by sudden current spikes during testing, can lead to test failures and unreliable results. Addressing voltage droop is essential to ensure the robustness and accuracy of the testing process, especially in scenarios involving Design for Testability (DfT) hardware or concurrent operations like parallel MBIST activations. This section introduces innovative techniques specifically designed to mitigate voltage droop during testing, focusing on simple, cost-effective solutions that leverage predefined test flows to maintain system reliability without significant overhead.

To understand the impact of voltage droops, Fig. 2.11 illustrates how they can lead to test failures. A fail arises when the voltage droops below a value called V_{fail}. In order to avoid unexpected failures due to voltage droop, we devised a voltage compensation technique. Our compensation technique requires three steps:

1. Identify the critical test steps. These steps cause a current spike and are not generated by the Circuit Under Test (CUT) but by its DfT or other parts of the DUT. These current spikes generate dangerous voltage droops that can cause the DUT to fail.

2. Raise the supply voltage of the system just before the critical steps. The voltage is raised enough to prevent a fail induced by the drop.

3. Lower the supply voltage to the starting value once the test has passed the critical step. In this way, it is possible to test the CUT at the minimum voltage set by the manufacturer.

Table 3.1 compares our proposed approaches with other solutions [36][37][38][39], analyzing their differences. Our proposals have the lower complexity with respect to all the analyzed solutions, requiring almost no extra components. Similarly to the approach in [36], the proposed approaches only works with known droops. This is not an issue for Automotive SoCs tested with a predefined test flows. The method proposed in [36] requires a relatively big transistor charged to an higher voltage than the supply of the circuit. This higher voltage capacitor is then used to inject current just when needed. Of course there is the added complexity of charging the capacitor to an higher supply voltage through another voltage source that needs to be disconnected when the current injection is needed. The work shown in [37][38] and [39] are part of a modified DC-DC converter. These methods do not require previous knowledge of voltage droop timings but require the devising and building a new kind of DC-DC converter. In the testing scenario, the sequence of tests is predefined by the manufacturer, so our proposed approaches leverage this knowledge to create two circuits that are as simple, cheap, and reliable as possible.

The first proposed approach utilizes an Embedded Voltage Regulator (EVR) to adjust the supply voltage during critical test steps dynamically. For devices without an EVR, we present an alternative solution based on external circuitry, which employs a Schottky diode and an NMOS transistor to achieve the desired voltage shift. Both approaches are designed to perform gradual voltage adjustments, avoiding abrupt changes that could cause dangerous current spikes.

Table 3.1: Comparison of the proposed approach with other Voltage droops mitigation techniques

	Masilamani et al.	Barrado et al.	Boscaino et al.	Amoroso et al.	Proposed
Droop Mitigation Technique	Non Linear Control Loop	Alternative DC-DC converter	Alternative Control System for multi-phase DC-DC converter	Auxiliary overcharged Decoupling capacitors	Supply Voltage Shift
Activation Mechanism	Automatic Compensation (Integrated in a DC-DC converter)	Automatic Compensation (Integrated in a DC-DC converter)	Automatic Compensation (Integrated in a DC-DC converter)	Externally Controlled	Externally Controlled
Voltage droop type	Any	Any	Any	Known Droops	Known Droops
Area Overhead & Complexity	Medium	Medium	Medium	Medium	Low (four components)

In the proposed approaches, shifting the supply voltage consists of 4 steps:

1. Raise the voltage to an higher level when a critical step, is about to start.
2. Wait for the voltage to reach the set value and continue the test.
3. At the end of the critical step, lower the voltage again to V_{min}.
4. Wait for the voltage to reach the set value and continue the test.

3.2.1 Voltage Droop During Testing

The first phase of our proposed approach is the empirical search of critical steps that generate consistent voltage droops. To find these critical steps, the evolution of the Voltage-Current behavior is monitored along the testing flow of the DUT. The instruments involved in this search are oscilloscopes equipped with both voltage and current probes or a tester with analog capabilities. In addition, to help this analysis, a GPIO of the DUT is also used as a fail/pass signal. The GPIO is particularly useful to understand which droops translate in test fail. Current spikes and corresponding Voltage droop are identified, and the specific embedded components that generate them.

Fig. 3.6 shows an example of a voltage droop causing an integrated voltage monitor to raise a fail signal. After the signal is raised, the test is interrupted abruptly. In the particular example shown in Fig. 3.6, the voltage droop is caused by a CPU exiting the idle mode to check the state of an MBIST (causing the test to fail).

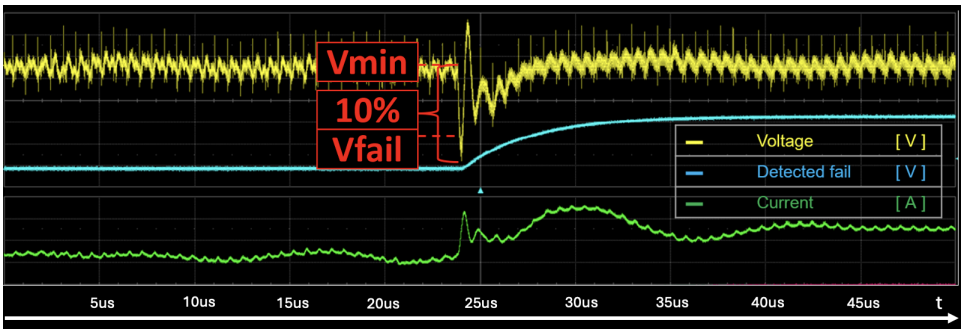


Figure 3.6: Current spike generates a Voltage droop and a fail signal raised by a GPIO

3.2.2 Proposed Mitigation Techniques

Once the critical steps are found, the tests are repeated with our devised countermeasure. When reaching the critical part of the test discovered in the previous step, the supply voltage is shifted to a slightly higher value. The first proposed approach uses an Embedded Voltage Regulator (EVR) to reach this goal. For devices that do not include an EVR, a solution based on external circuitry is also proposed.

A sudden shift in the supply voltage would cause dangerous circuit current spikes due to the supply lines' parasitic components. To avoid these current spikes, our approaches are used to "slowly" raise the voltage to the chosen safe value.

Embedded Voltage Regulator

The EVR is organized as in Fig. 3.7. Once the target voltage is set, a digital controller drives the gates of two MOSFETs (one of type P and one of type N). The drain of the PMOS is connected to an external power supply providing "V_{ext}", while the source of the NMOS is directly connected to the ground. An inductor is connected to the node between the PMOS source and the NMOS drain, followed by a grounded capacitor. These two reactive components help the EVR to have a stable and glitch-less output.

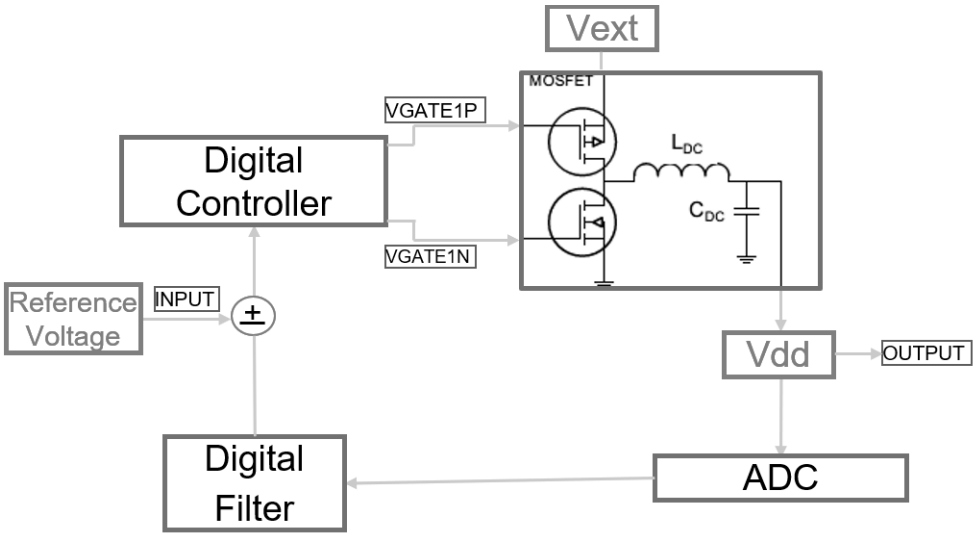


Figure 3.7: Embedded Voltage Regulator schematic view

The EVR uses an internal voltage reference to decide how to drive the two MOSFETs and consequently set the output voltage. The output voltage is compared against the reference one and set to the desired value. Depending on the result of this comparison, the digital controller decides the changes needed for the input of the two gates. For example, if the output voltage is too high with respect to the reference one, the voltage of the NMOS is increased, together with the voltage of the PMOS, and vice-versa.

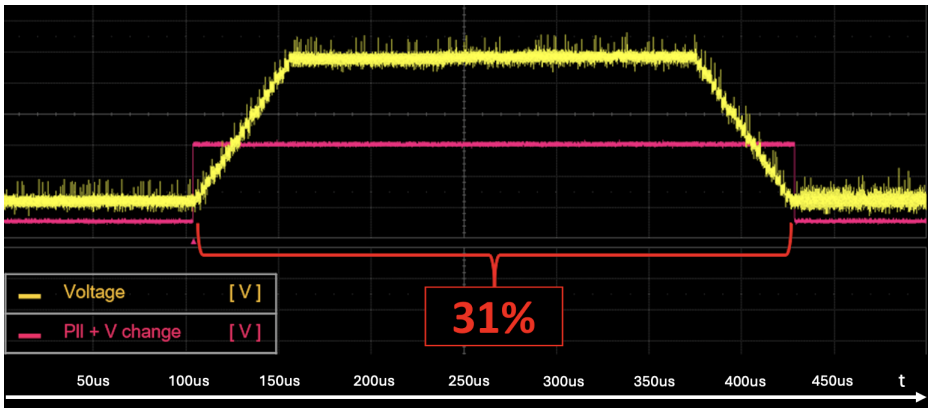


Figure 3.8: Effect of voltage droop at different supply voltage

The EVR is programmed to perform the voltage shift in small steps. The small shifts performed one at a time help reduce the current spikes created by a sudden shift in the supply voltage. A visual representation of this slow shift is shown in Fig. 3.8.

External circuitry-based approach

The second proposed approach uses external circuitry to achieve the desired voltage shift. This approach is best suited for DUT that does not include an EVR but still needs a technique for voltage droop compensation.

The circuit, as shown in Fig. 3.9, has a Shottky diode and an NMOS connected in parallel between the power supply ground and the ground of the DUT. A Shottky diode has been used for this implementation to have a 0.3V voltage droop, but a diode with a higher drop can be used to have a lower V_{min}.

The output of a low-pass filter is used to drive the NMOS gate. The low pass filter is connected to one of the DUT's GPIOs allowing the DUT to turn on and off the transistor. During normal operation at V_{min}, the transistor is shut off, setting

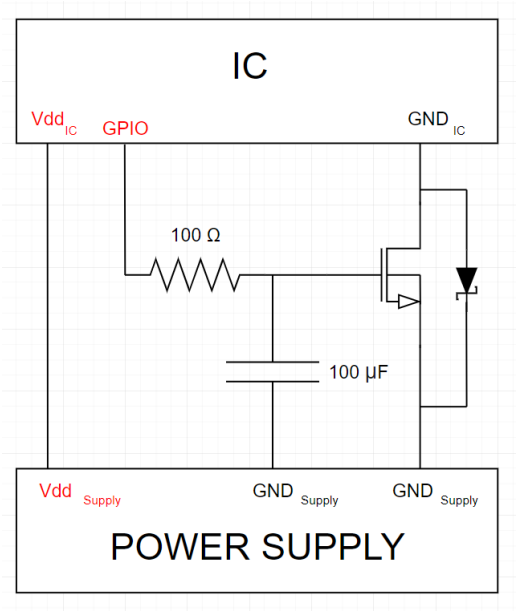


Figure 3.9: Diode and Mos setup

the GPIO to a low output value to have the diode as the only path between the two grounds.

In a first approximation, the Schottky diode can be seen as a $V_{Schottky}$ Voltage generator as seen in Equation 3.1

$$GND_{\mathrm{DUT}} = GND_{\mathrm{PowerSupply}} + V_{\mathrm{Schottky}} \tag{3.1}$$

From this relation, it is clear that the two grounds are not equipotential and that the DUT is effectively receiving the Voltage set in the power supply minus $V_{Schottky}$ as in Equation 3.2.

$$V_{\mathrm{DUT}} = V_{\mathrm{PowerSupply}} - V_{\mathrm{Schottky}} \tag{3.2}$$

Consequently, to perform a test at V_{min}, the power supply is set as shown in Equation 3.3.

$$V_{\text{PowerSupply}} = V_{\text{min}} + V_{\text{Schottky}} \tag{3.3}$$

During critical steps, the transistor is switched on, setting the GPIO to a high voltage value. A transistor can be considered a simple switch in a first approximation. When turned on, the transistor provides a low resistance path between the two grounds, effectively eliminating the effect of the diode as seen in Equation 3.4

$$GND_{\text{PowerSupply}} = GND_{\text{DUT}} \tag{3.4}$$

Consequently, the effective voltage supplied to the DUT is expressed in 3.5.

$$V_{\text{PowerSupply}} = V_{\text{DUT}} = V_{\text{min}} + V_{\text{Schottky}} \tag{3.5}$$

It is important however not to change the voltage too abruptly, as this would generates dangerous current spikes.

To understand how we can slow down the voltage change, it is necessary to take into consideration the NMOSFET Voltage-Current relationship. As the I_{DS} of the transistor is related to the V_{GS}, it can be considered a variable resistance voltage-controlled. The value of this resistance goes from infinite (Cut Off region) to R_{On} of the transistor, which usually has a value in the order of the mΩ. In order to slow down the voltage shift and allow a smooth current increase, the effective resistance of the transistor is changed by using a low pass filter to modify the V_{GS} gradually.

As known, the τ of a low pass filter can be expressed as shown in 3.6.

$$\tau = R * C \tag{3.6}$$

When turning on the GPIO used as input for our filter, it is possible to consider that at 4τ, the capacitor is fully charged to V_{GPIO}. The GPIO's High state voltage is not necessarily high enough to bring the transistor into the saturation region. However, it should be enough to obtain the desired voltage shift. Remembering the $V_{PowerSupply}$ we can obtain the ranges shown in the Double inequality 3.7 for V_{DUT}.

$$V_{\text{min}} \geq V_{\text{DUT}} \leq V_{\text{min}} + V_{\text{Schottky}} \tag{3.7}$$

ADC Feedback

The approaches that have been studied need a certain amount of time to perform the voltage shift. Multiple ADC reads are performed to ensure that the critical step is executed only when the correct supply voltage is applied. Through these reads, the CPU is stalled until the correct value is detected. The same check is performed again once the supply voltage is shifted back to its original value after the execution of the critical step.

Methods comparison

Both the proposed methods are effective in compensating voltage droops during SoCs' testing but have their characteristics, and the best method to use is strongly dependent on the SoC itself. The EVR-based method is generally superior to the external circuitry-based approach, but not all SoC have an EVR unit available. Table 3.2 summarizes the differences between the two approaches.

Table 3.2: Comparison between the proposed approaches

Compensation Method	Compatibility	Voltage Shift Timing	Voltage Shift Amount	Circuit Complexity and Cost
EVR	SoC with available EVR + ADC	Programmable	Programmable	Low 4 components
External Circuitry	SoC with available GPIO + ADC	Fixed by the low pass filter	Fixed by the diode voltage droop	Low 4 components

To validate the proposed approaches, we used an Aurix^{TM} TC49 made by Infineon^{TM} that already contains an EVR. A photo of the experimental setup is provided in Fig. 3.10 that shows the board with the DUT and the oscilloscope used to measure the Voltage and Current signals. Contrary to the method proposed in [36], which requires the power supply to provide two differents voltages to overcharge a capacitor, the proposed methods just use a single power supply's channel. The proposed approaches perform a voltage shift independently from the power supply.

During our analysis with this DUT, we discovered that one of the most critical steps was related to a parallel MBISTs activation step. This parallel MBISTs activation generated consistent current spikes able to cause a dangerous voltage droop. The voltage droop was particularly critical when performing a memory test at Vdd min.

In accordance with Infineon disclosure agreement, the following results will exclusively show relative percentages and not absolute values.

Finding critical steps

The proposed approaches require knowledge of the so-called critical steps. In this context, we defined critical steps as the test steps generating a sudden current spike and, consequently, a dangerous voltage droop.

To find the critical steps in a particular test flow, all the flow is executed without any voltage droop compensation technique in place. During the flow execution, the voltage and current behavior is monitored to search for dangerous current spikes

Fig. 3.11 shows the test scenario without our voltage droop compensation step. The fail signal is raised due to the voltage droop, effectively causing a failed test.

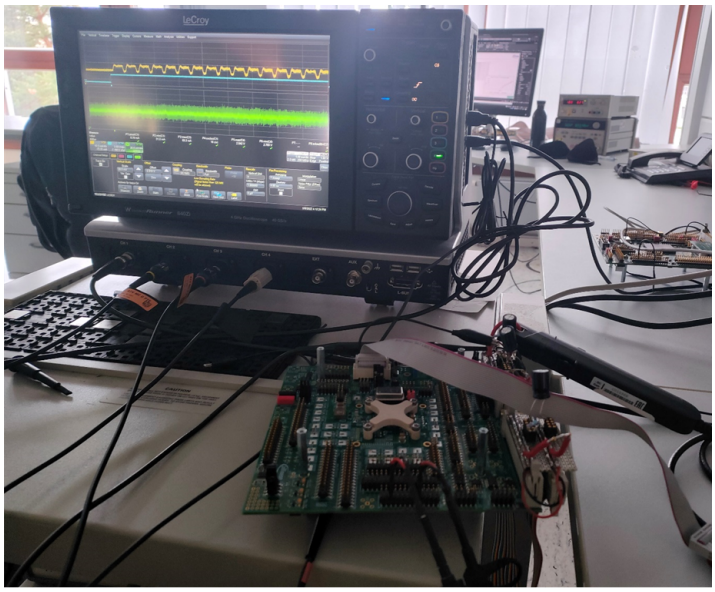


Figure 3.10: Experimental setup with board and oscilloscope

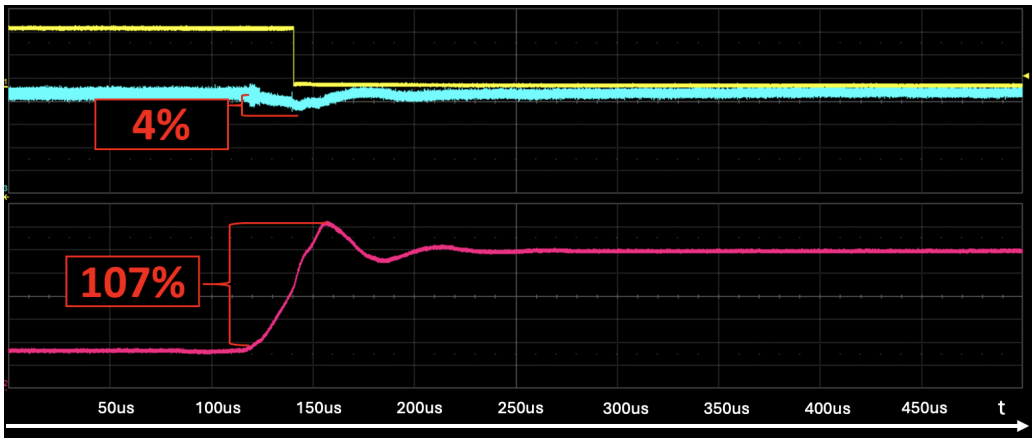


Figure 3.11: Current Spike and Voltage Droop during a parallel MBISTs activation step

Supply voltage shift

The first experiments will show the behavior of the DUT current and voltage when performing a voltage shift in both sudden and gradual shifts in case of an idle device. After these initial considerations we will show how knowing the critical test step our compensation approach can be used to effectively prevent dangerous current spike and consequently voltage droop.

Fig. 3.12 shows the behavior of our system when the supply voltage is raised by 11% in a single step using the integrated EVR. The sudden change in the supply

voltage generates a dangerous current spike that peaks at 470%, the nominal value. A current spike of that amplitude can be harmful to the DUT, even if it lasts for a small amount of time.

The solution for the current spike is to program the EVR to raise the supply voltage in a sequence of small steps. The resulting scenario can be observed on Fig. 3.13. As seen at every small step, a well-tolerated small current spike of approximately 50% is generated (the figure use the same scale as Fig. 3.12).

Fig. 3.13 shows the same increase of 11% of the supply voltage with respect to the nominal value but this time performed in 20 small steps of 0.55% each. Fig. 3.13 clearly show that performing the voltage shift in smaller steps is more efficient in terms of current spikes. Performing the voltage shift step by step, the current peaks decrease by 420% with respect to the nominal value. The drawback of this approach is the impact on the timing, the component that causes more problems is the ADC reading that is done two times, every time the voltage change thanks to the EVR. Fig. 3.14 shows the system's behavior when the MBISTs are concurrently activated and the voltage is subsequently shifted down with a series of small steps. As shown in the figure, the peak due to the MBISTs' activation now lies in the "safe" voltage zone, and the subsequent voltage shift generates just small, and not dangerous, peaks.

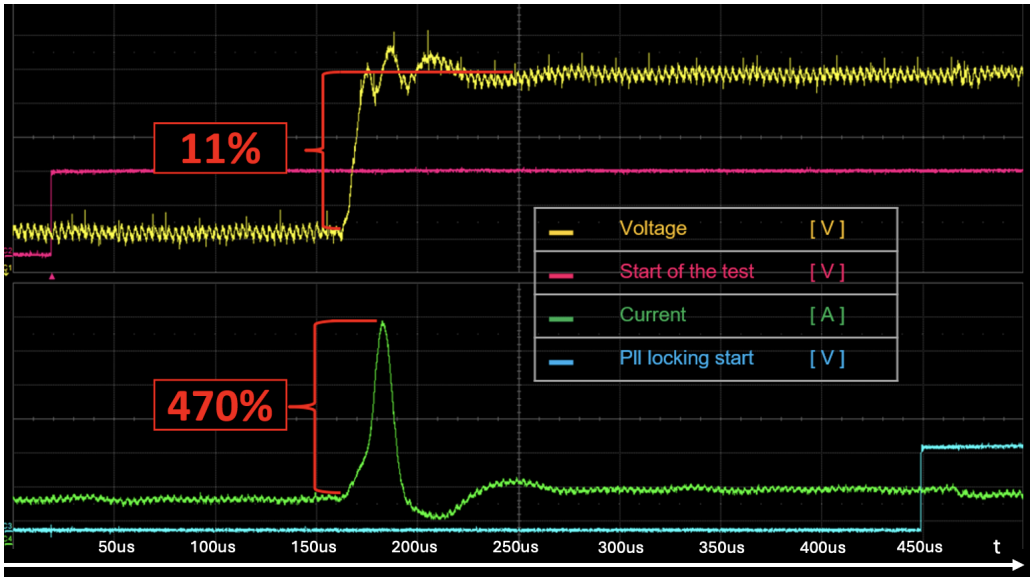


Figure 3.12: Current behavior with sudden supply voltage shift

Fig. 3.15 shows the behavior on the current of the DUT in case of a sudden shift of the supply voltage through the proposed external circuitry. The figure refers to the circuit proposed on Fig. 3.9, but without the 100uF capacitor and the 100 Ohm resistor to obtain the sudden change in the supply voltage. As can be seen, by raising the supply voltage by 22% a 214% current spike is generated. The

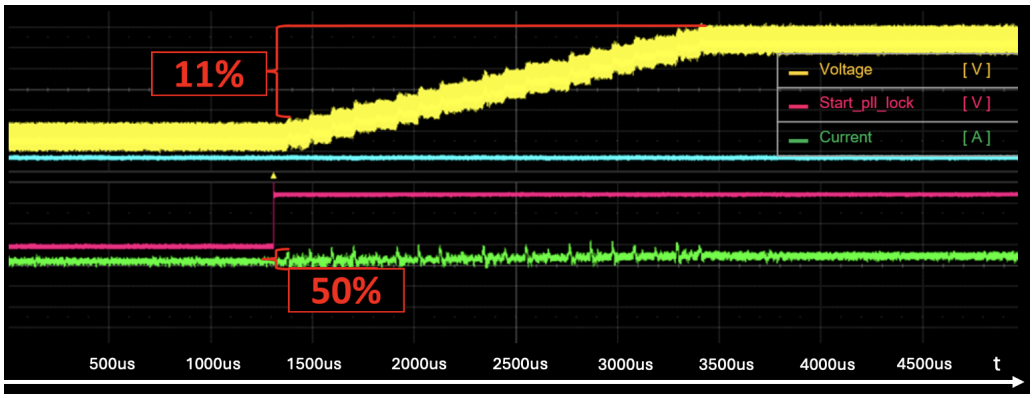


Figure 3.13: Current behavior using EVR for small voltage shift steps

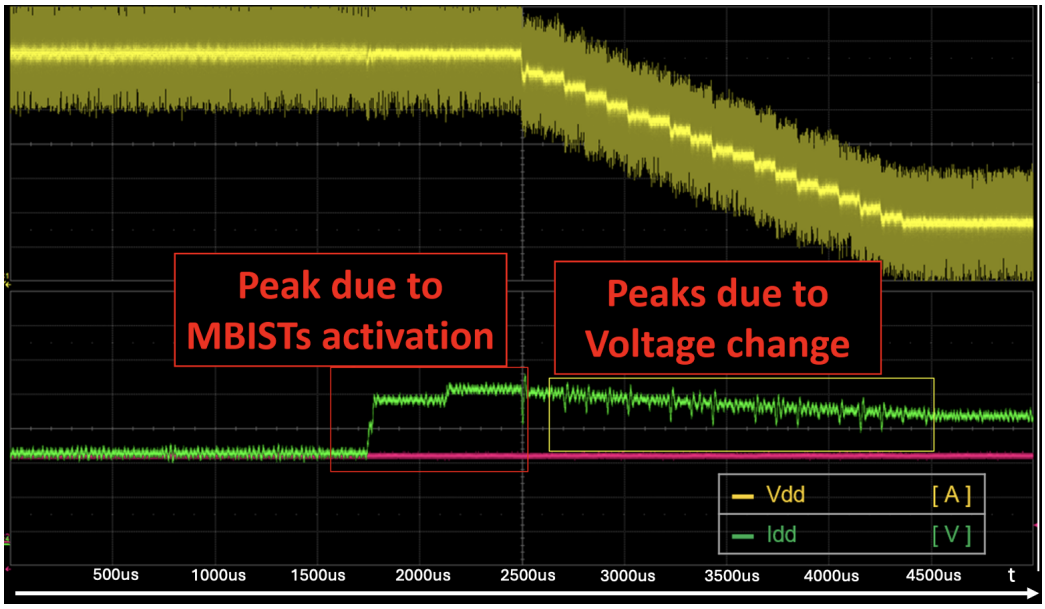


Figure 3.14: MBISTs activation phase with EVR setup

current spike is lower than the one obtained through the EVR because the intrinsic capacitance of the Mosfet is by itself delaying the saturation of the transistor and providing a small delay in the execution of the voltage shift. So also in this case it is necessary to slow down the voltage shift to prevent dangerous current spikes. Fig. 3.16 shows the behavior of the external circuit with the addition of the 100uF capacitor and the 100resistor. Again in this case the supply voltage is raised by 22%. In this case there are no discernible current spikes and the current increase just by 81% with respect to the nominal value as an effect of the increased voltage.

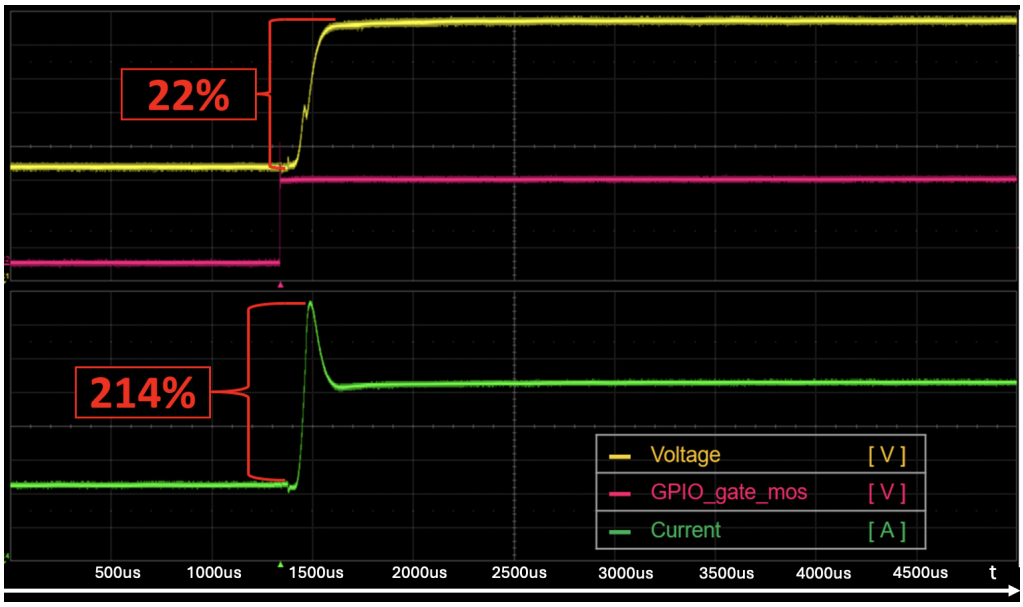


Figure 3.15: Current peak problem with diode and mos setup

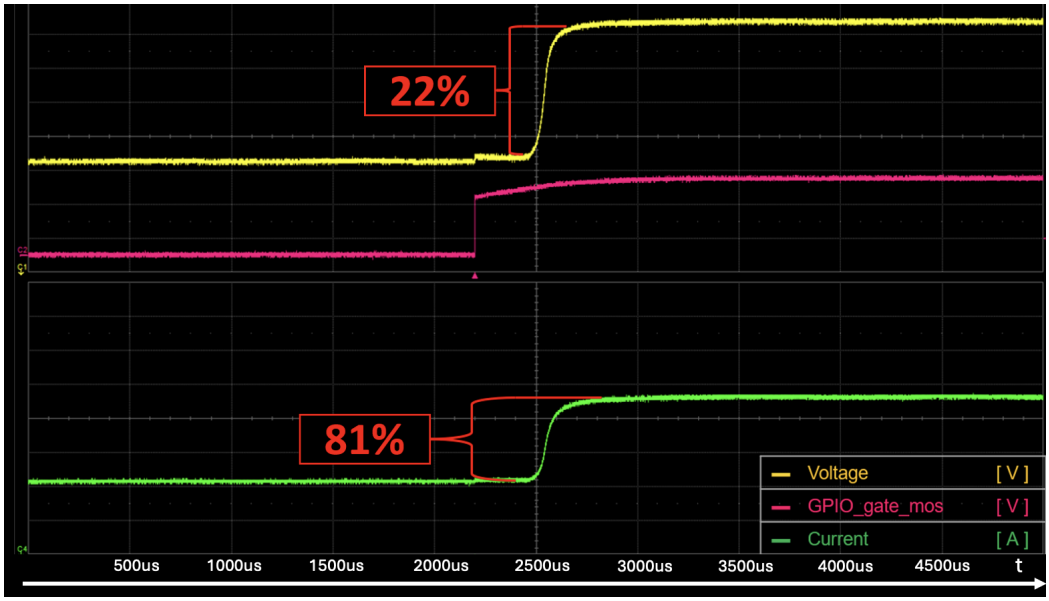


Figure 3.16: Results with diode and mos setup

Timing

In this section, the time overhead of the proposed EVR approach is analyzed. In the following pictures, the focus is on the EVR approach as both approaches showed similar timing and as the most time overhead is introduced by the ADC readings anyway. Both the sudden and gradual supply voltage shifts are analyzed

to see the difference. Fig. 3.17 shows the time overhead when the supply voltage suddenly shifts to the desired voltage and then suddenly returns to the previous one. As can be seen, the overhead with respect to the parallel MBISTs activation operation, in this case, is just 5.5%. Still, it also shows the dangerous current peaks discussed in the previous section.

Fig. 3.18 shows the time overhead of the voltage shift performed with a series of small steps. In this case there is an overhead of 25.5% if the voltage change is done in 20 0.55% voltage steps to abtain the 11% shift in the supply voltage.

Fig. 3.19 also include the time overhead introduced by the ADC readings, integral part of the proposed approach. The proposed approach requires two ADC readings, performed when the voltage is raised to the "safe" level and when the voltage is lowered back to its original value. By taking in consideration the two ADC reading the time overhead increase to 800 % with respect to the parallel MBISTs activation timing. Of course in the overall test time, an 800% time overhead on a single (and relatively fast) test step does not cause a sensible difference, but it is incredibly valuable to prevents fails due to voltage droops.

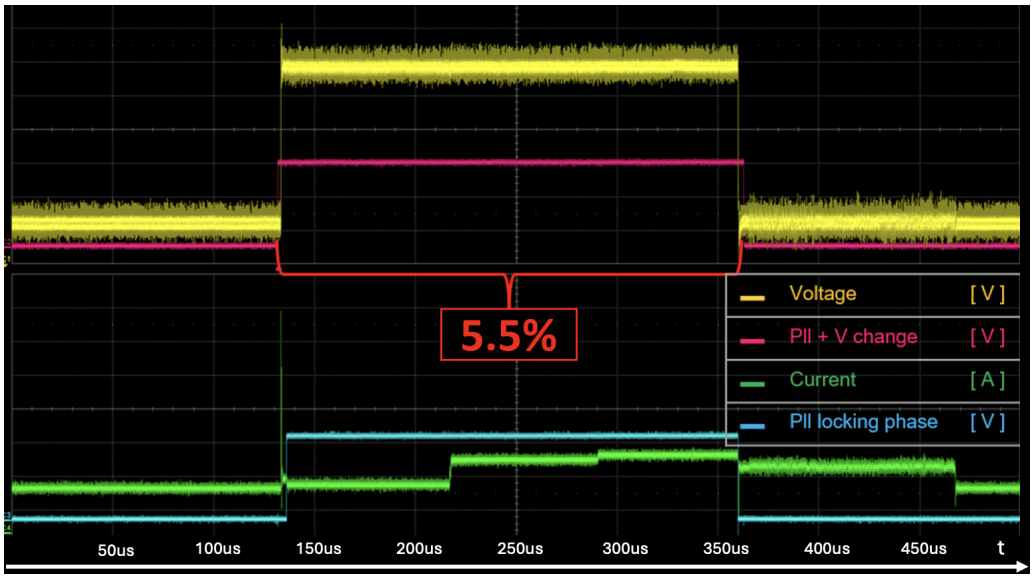


Figure 3.17: Timing with 11% voltage change in one step during parallel MBISTs activation

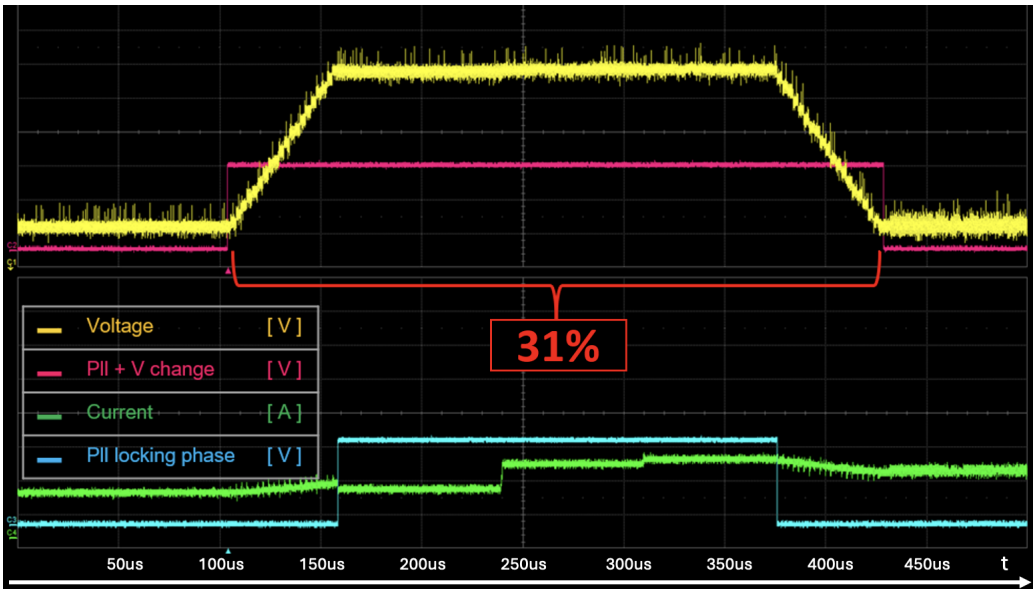


Figure 3.18: Timing with 11% voltage change in 0.55% steps during parallel MBISTs activation

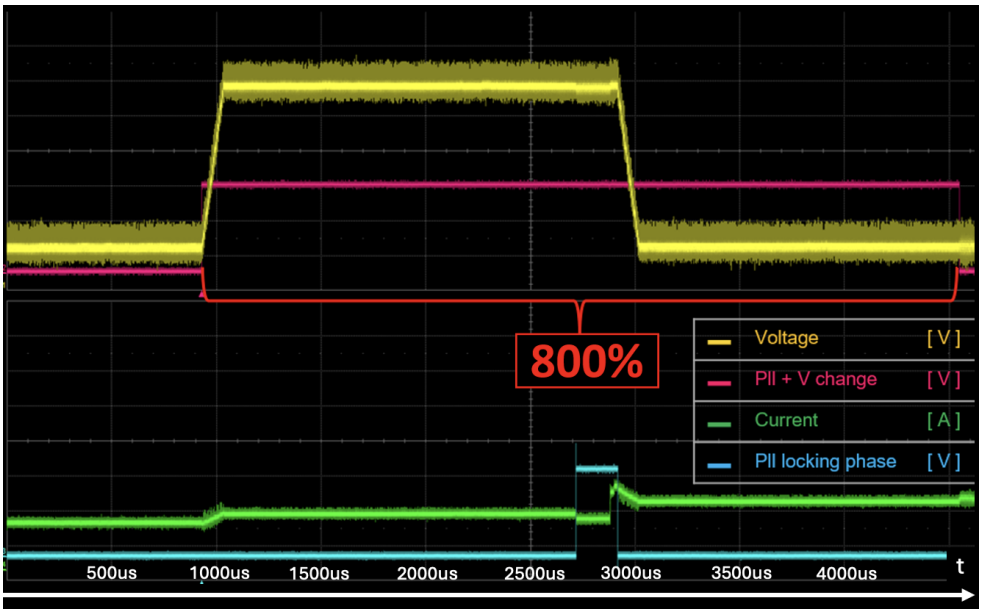


Figure 3.19: Timing with 11% voltage change in 0.55% steps and adc read during parallel MBISTs activation

These timing overheads are negligible in our test setup; since our compensation only happens in critical and limited test steps, the total time overhead in the test flow was less than 0.01% of the total test time.

3.3 Optimized Diagnostic Data Compaction and Compression

In modern semiconductor testing, the ability to efficiently store and analyze diagnostic data is critical for ensuring the reliability and performance of devices under test (DUTs). Embedded memories, such as Flash, are particularly challenging due to their high density and the need for detailed failure analysis. Traditional diagnostic methods often face limitations in terms of memory usage, accuracy, and on-chip implementation feasibility, especially when dealing with fault-dense scenarios.

This section introduces a series of innovative algorithms and methodologies aimed at optimizing diagnostic data compaction and compression. These approaches address the challenges of limited on-chip memory, test time constraints, and the need for accurate fault representation. The proposed methods leverage encoding, compaction, and compression techniques to maximize the diagnostic information stored on-chip while minimizing memory requirements and computational overhead.

The section is structured as follows:

- **Fault Shape-based Encoding Strategy**: A novel on-the-fly encoding method that compacts failure bitmaps into "colored slices" for efficient storage and analysis. This work has been published at the IEEE ETS 2022 conference [24]
- **Delta-Oriented Compaction Algorithm**: An enhancement to the encoding strategy that focuses on storing only the differences between successive tests, further reducing memory usage. This work has been accepted at the DATE 2025 conference as a Late Breaking Result.
- **Density-Oriented Compression Algorithm**: A pixel-based compression approach that represents fault density across memory regions, enabling efficient visualization and analysis of fault patterns. This work has been published at the IEEE ETS 2023 conference [25].

By combining these techniques, the proposed solutions provide a scalable and efficient framework for diagnostic data management, particularly suited for automotive and safety-critical applications.

3.3.1 Fault Shape-based Encoding Strategy

The proposed approach is based on the concept of encoding to create on-chip compact failure bitmaps. By leveraging on a composite test architecture, the

Table 3.3: Comparison between capabilities of various methodologies

Method	Landzberg[2]	Chen [16]	Proposed
Accuracy	Complete	Partial	Complete
Storage	Low	Variable	Very Low
On-chip	Yes	Possible	Yes

bitmap information is stored in encoded or "colored" segments that we call "slices" and updated along with the execution of the test.

The proposed compaction approach guarantees high accuracy, similarly to [2]. Conversely, [16] returns an approximated information due to compression as summarized in Table 3.3. About memory needs, the proposed method requires fewer memory resources than required by [2], and slightly more than the method in [16] when the minimum compression ratio is used. The proposed method run on-chip enabling the download of complete information at the end of the test, as it is done by [2] and possibly with the method in [16], which is originally implemented by additional hardware and tester capabilities. The bitmapping schema we present is

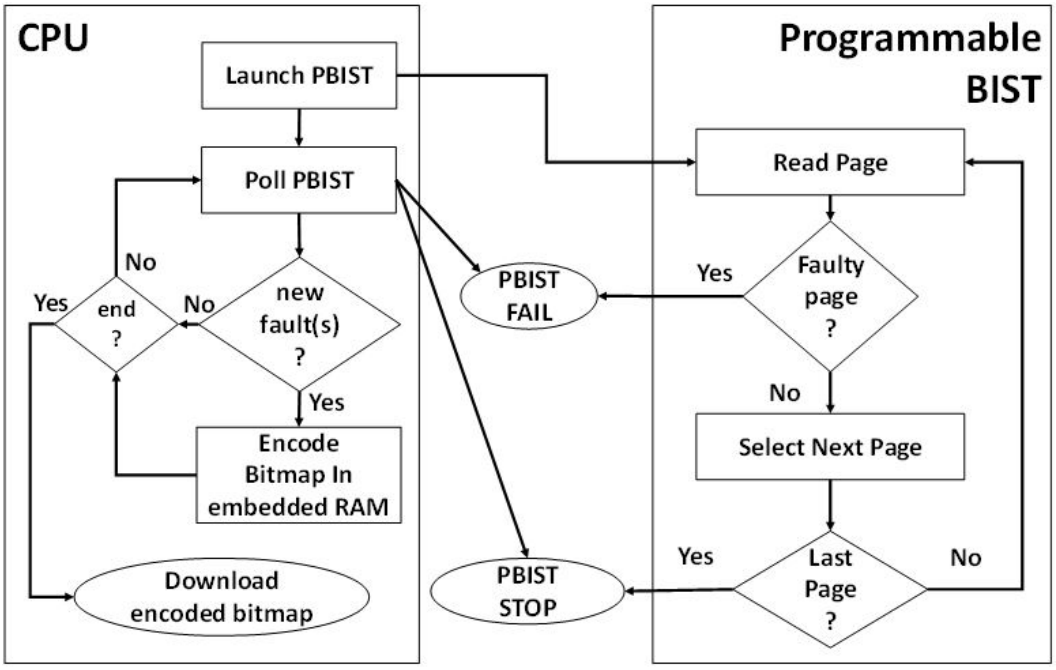


Figure 3.20: CPU and Programmable BIST organization

supported by a suitable hardware design, where a programmable BIST is directly accessible from a CPU like in [40] and [41]. A sketch of how the Flash memory design for test works is in Figure 3.20. The CPU activates the selected procedure to be run by the Programmable BIST and then waits for the failure events. When

a fault is encountered, the BIST stops and raises a flag. Once the CPU notice this flag through polling actions, it can access the data, resume the BIST operation and then perform some computations. These on-chip computations may involve a repair algorithm that allocates some redundancy elements and a bitmapping algorithms like the one described in this thesis. In Figure 3.21, a Golden fault-free test execution is shown. Here, after the Init phase, the BIST independently tests the entire embedded Flash memory in a reference time called "t_{gold} ".

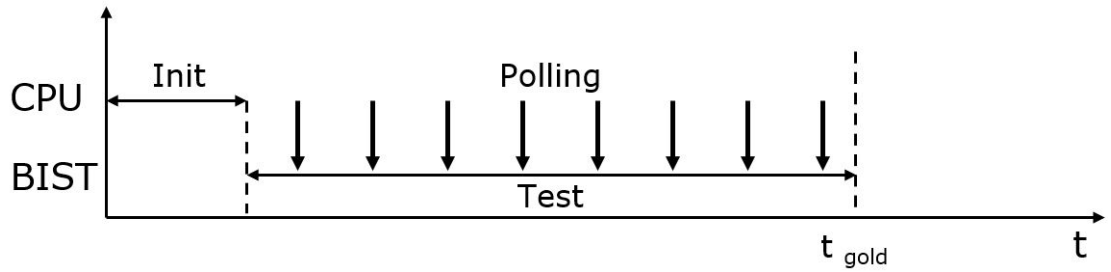


Figure 3.21: Golden test execution

In Figure 3.22 a different scenario is shown. Here the BIST finds a fault and stops, waiting for the CPU to read it and resume its operations. At this point, the CPU and the BIST can work independently in an interleaved fashion. So, while the BIST is busy testing the rest of the memory, the CPU can analyze the discovered fault and run a bitmapping algorithm or the coloring algorithm proposed in this thesis.

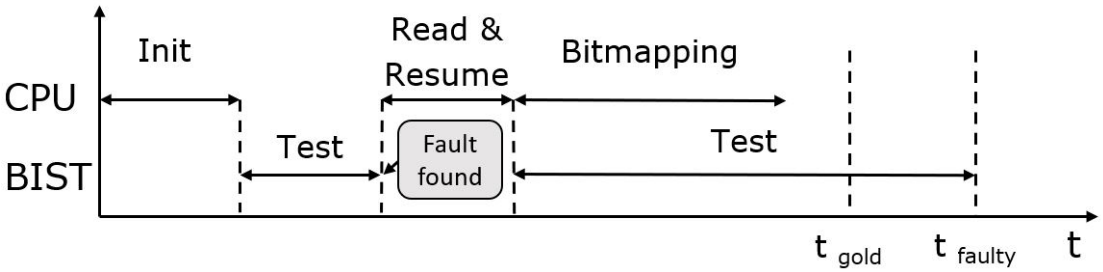


Figure 3.22: Test with a faulty bit and interleaved CPU & BIST operations

The total test time increase, and it is now "t_{faulty}", which is less than the sum of the single time components of the entire system (e.g., t_{faulty} is smaller than the sum of t_{gold}, t_{read} and $t_{encoding}$). Such a folding method is very beneficial for saving test time when a computation is required to react to a fault appearance. In our case, we exploit this possibility to encode the bitmap information stored on-chip incrementally. Every time the PBIST returns a fault, the encoding algorithm is executed, and the current bitmap information is updated.

Proposed encoding strategy

The goal of the proposed approach is to produce an on-chip and on-the-fly encoded representation of the failing bitmap. The primary objective of the method is to maximize the number of information that can fit into a pre-assigned amount of on-chip memory. The on-chip memory constitutes a very strong constraint. Suppose the available memory resource is exhausted before the test has ended. In that case, the bitmap will result incomplete, or the tester should intervene by downloading the current part, resuming the test, and continuing iteratively till the test ends. Despite the multiple download solution looks feasible in theory, few tester architectures support it, and it heavily impacts on the test time.

Therefore, the most viable solution to save a more significant number of complete bitmaps is to compact them by encoding the information. The trade-off is with the test time overhead introduced by the encoding computations.

In our methodology, we encode on-chip and on-the-fly the bitmap information into "colored segments", also called "slices", which are the basic structures of our compaction algorithm. We chose segments over other kinds of shapes (i.e., rectangles) after a careful examination of thousands of failure constellations. Fails mostly align over wordlines and bitlines, making segments the most efficient and straightforward way to encode them. A slice represents one or multiple faults belonging to the same bit-line or word-line, in a format that includes:

- The indication if the segment is horizontal or vertical
- The part of the physical coordinate of the first and the last faults in the segment
- A color, which characterizes the segment in consideration of the distribution of the faults it covers.

For our approach, we propose four colors as depicted in Figure 3.23 and described below:

[label=)]Black: a black segment includes one single fault Blue: represents two faults far from each other Red: represents two or more faults at odd or even positions (one or more faults regularly interleaved by working bits). This color is beneficial when a checkerboard pattern is applied, as memory is tested exactly in its encoded pattern. Orange: Two or more faults physically adjacent

In Figure 3.23, the left-hand part shows the actual bitmap, while on the right, it reports the colored segments or slices.

The purpose of the proposed approach is to create a set of slices that fit the failure constellations of our DUTs. Such a set is built on the fly by the CPU that

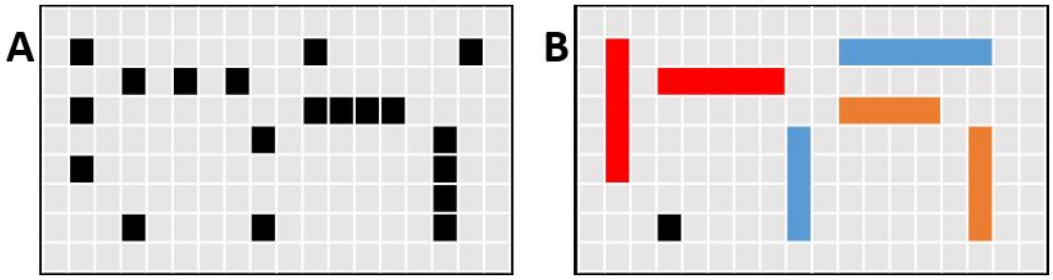


Figure 3.23: Fault shape to color representation

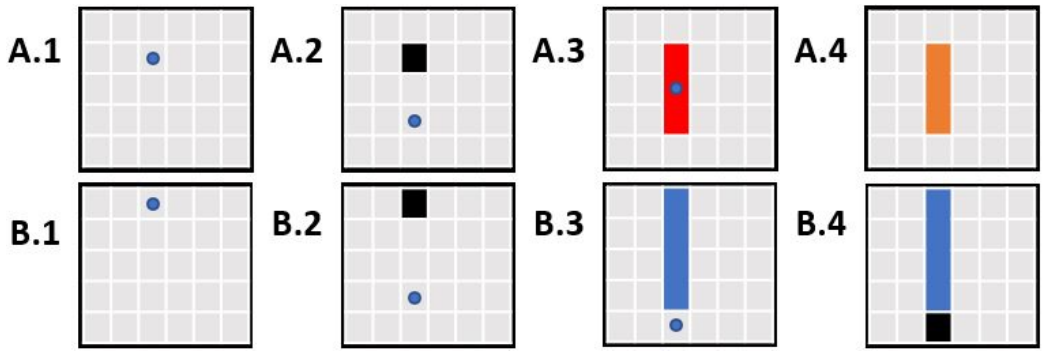


Figure 3.24: Slice being updated based on the position of the new incoming fault represented as blue dots

reacts to a new fault arrival by updating the existing slice's content or by initializing a new one.

In Figure 3.24, an example of how slices can be updated when new faults arrive is shown:

3. At first, a fault is received, A.1, and a black slice is created in A.2. In A.2, we also receive a new fault that causes the update of the black slice to the red slice in A.3. Similarly, the fault received in A.3 in the middle of the red slice causes its update to an orange slice shown in A.4

- At first, a fault is received, B.1, and a black slice is created in B.2. In B.2, we also receive a new fault that causes the update of the black slice to the blue slice in B.3. In B.3, a fault is found just below the blue slice. Since the blue slice cannot contain an additional fault, a black slice is created to encode this last fault in B.4

On-chip memorization of the encoded information

An important issue to consider is how the on-chip memory is organized. This is not only relevant in terms of potential storage capability but also in terms of

access time to the memorized information. In fact, the algorithm should be able to quickly check the already included information to evolve the current encoded bitmap. In other words, the algorithm must search in the current set of slices if there is one of the existing to update or to create a new black one.

The approach we adopted and illustrated targets both the minimization of the information to store and the time required by the algorithm to process a new fault. The memory organization is like the one used in caches, and it implements a set-associative approach.

Given the number N of selected sets, the available memory is split into N equal parts. When a new fault is logged, its address and fault mask are retrieved by the CPU, which manipulated them to extract three components:

- From the word-line address it derives
  - The set index which the slice belongs to as, computing Address%N
  - The coordinate of the fault normalized by the set calculated as Address/N
- From the fault mask, it extracts a tag that is then used to perform the searches, indicating the bit's position in the fault mask.

In Figure 3.48 illustrates with an example how the PBIST output is parsed.

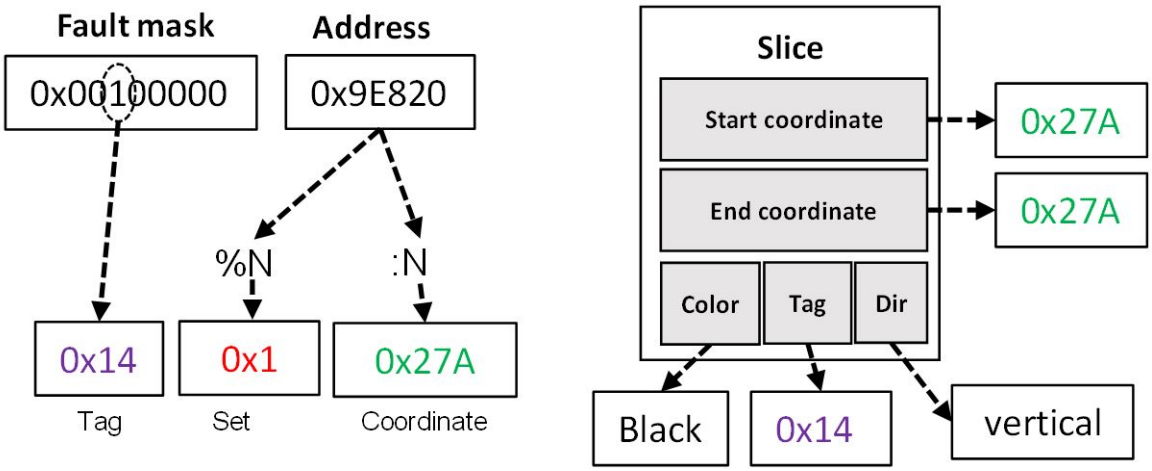


Figure 3.25: Fault information analyzed by the CPU when N=32

Figure 3.49 completes the memory organization overview, in a case where the number of sets is N=32, and the fault mask includes 256bits. As dictated by the set-associative organization, the memory is divided into N chunks of equal size. Once the set is computed out of the fault information, the correct memory portion is accessed, and the tag is used to look in the set for a slice with the same tag

value. If such a slice already exists in the corresponding set, it is manipulated as described before. Conversely, a new slice is stored if the current fault cannot be linked to any previously stored one.

The illustrated method is both very efficient in terms of search time and in terms of required bits. The division in sets permits the reduction of search time by a factor that depends on the number N of sets. The set value is not stored in the slice but can be inferred by reverse formulas from the slice address in the on-chip memory.

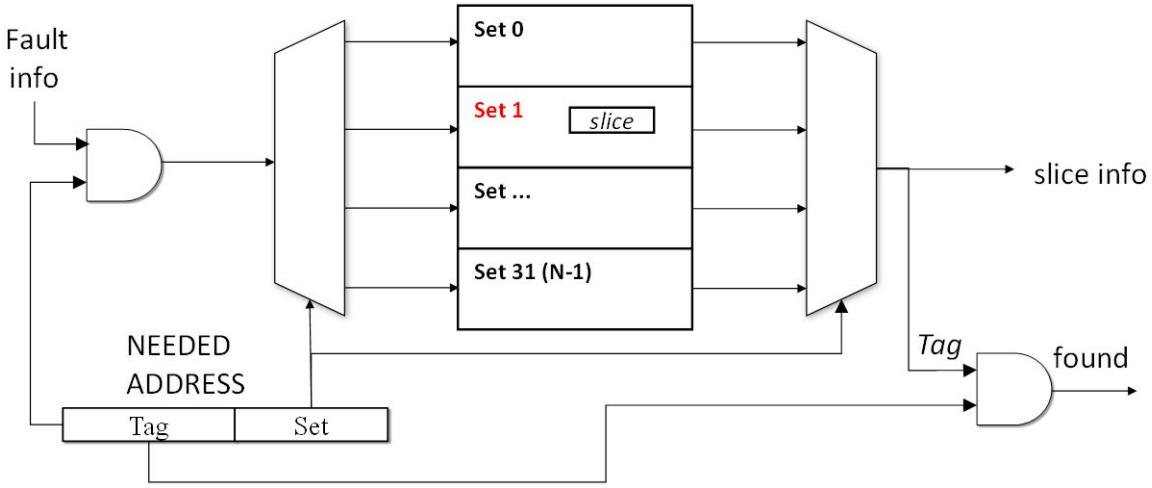


Figure 3.26: Cache-like organization of the available memory for on-chip slice storage

Selection of horizontal or vertical encoding direction

Of course, the fault mask may contain more than one failing bit. In such cases, the algorithm can create either a vertical (bit-line oriented) or a horizontal (word-line oriented) slice. Despite vertical coloring being easier and performed by considering a bit at a time in the fault mask, it is crucial to reduce its usage as much as possible by identifying horizontal shapes as soon as possible. To conclude that a segment is horizontally oriented is made difficult by the scrambling effects, which leads to the distribution of faults over many Flash memory pages in a word-line.

To solve the trade-off between speed and accuracy in the selection of vertical or horizontal direction, the horizontal coloring is triggered when the algorithm "guess" the presence of a horizontal shape. This guess is based on the number of faults received for the current page; the horizontal coloring is activated if it exceeds a given threshold.

The selection of direction mechanism is explained in Figure 3.50. Depending on the number of faults in the fault mask, they are either immediately colored one

by one vertically if their number is less than the threshold, or they are temporarily saved in a buffer for later coloring. In fact, to color horizontally is more efficient if all pages aligned on the same word-line according to the scrambling schema are processed together. Once the horizontal direction is taken, the temporary buffer is

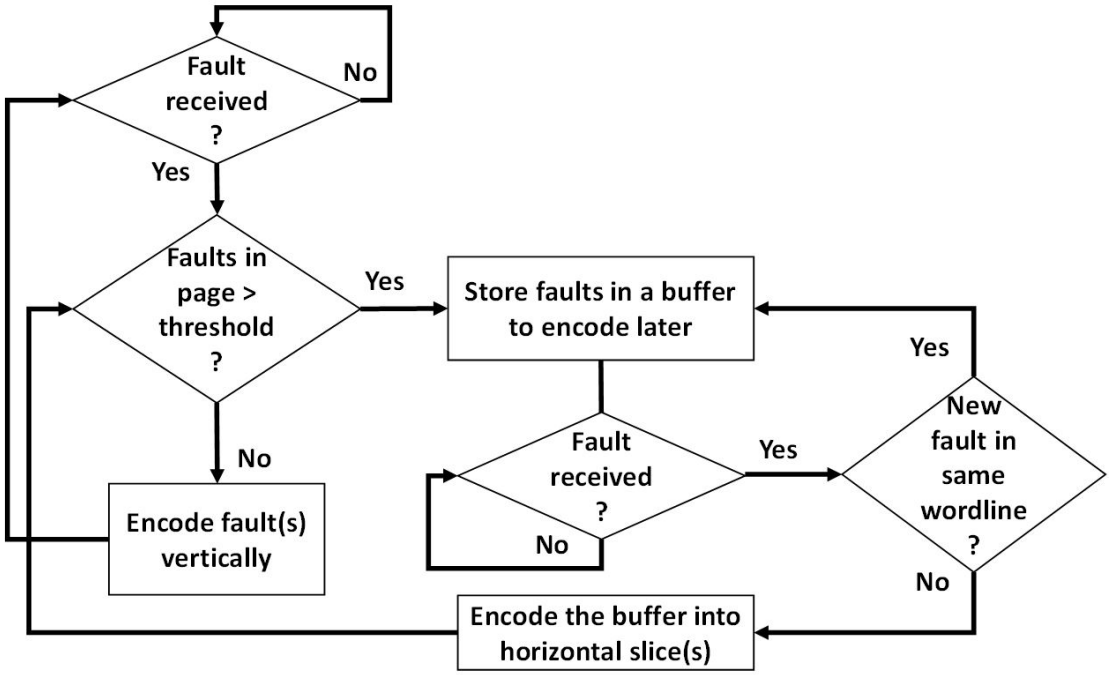


Figure 3.27: Flowchart of vertical/ horizontal encoding decision

updated with failure data eventually coming from other pages on the same word-line. The buffer content is processed at the first encountered fault that is no longer in the word-line under investigation. The created horizontal slice(s) is stored in the corresponding memory set.

Experimental Results

This section shows the results obtained by the proposed algorithm over various study cases. Our reference device is the Aurix TC39xB, manufactured by Infineon Technologies. For this device, a limit of 24KB in RAM memory to store bitmap information was defined during test operating system component sizing. The available on-chip memory constraint is the key factor to evaluate the approach.

In the following paragraphs, we will compare the pros and cons of the proposed compaction approach with respect to adopting a bit-by-bit coordinate-based approach [2] or a compression approach like in [16]. According to scrambling parameters, we divided such a space into 32 sets. The configuration parameters, including a fault mask on 256 bits and addresses on 32bits, lead to 6-bytes sized slices for the

proposed approach. Conversely, the bit-by-bit approach [2] directly saves the fault coordinates into 4-bytes elements and [16] resorts to shared bits among word lines and bit-lines.

Experimental results that follow show how the illustrated approach guarantees a solid average save in terms of memory requirements, meaning that it can store more information than the reference bit-by-bit approach [2] when the same amount of memory is supplied. In other words, the proposed method can fully log more failing devices than the bit-by-bit approach. The method pays a fee in terms of increased bitmap generation time, which looks sustainable given the occupation advantage.

About the comparison with [16], we were running experiments executed with a compression ratio of 4480x. This is the lowest resolution possible, and it requires a fixed 20KB of on-chip memory so that this method would be feasible on-chip with our memory limits.

At this point, we first report the advantages and costs of the proposed approach by referring to four real and typical fail scenarios coming from production data. For all of them, we report a comparison among the proposed approach and [2] that saves on-chip the complete list of failing cells coordinates.

Then, we consider a more extensive set of failing devices, around two thousand, which have been accurately selected to constitute a significant sample of the volume production. This part reveals that the bit-by-bit approach [2] is slightly faster but limited by the available on-chip memory space. At the same time, this limitation is lessened by the proposed approach. We also compared the average accuracy of bitmaps reconstructed after compression by [16] and computed a correlation index to grade the loss in accuracy with respect to our lossless method.

Fine analysis of some typical phases

The following figures are cropped bitmaps showing some specific regions of failing embedded Flash memories. Per every figure, there is (A) the failure bitmap and (B) the respective representation returned by our algorithm.

A vertical-oriented faulty scenario is shown in Figure 3.28. In this specific case, with the eFlash affected by 388 faults, the overhead with respect to the gold execution (test with a good memory) is 21ms. Compared with a bit-by-bit approach, which takes 14.35ms, our method shows a relative time overhead of 46%. Despite the time loss, the proposed algorithm was saving 95% of the required RAM space of about 78B with respect to the bit-by-bit approach requiring 1.51KB.

The vertical cases are those that fit more with the proposed algorithm. In fact, the PBIST takes some time to reach the subsequent fault and, therefore, the CPU takes great advantage of this time to perform the algorithm while PBIST is running.

In the case of Horizontal oriented shapes, like the ones in Figure 3.29, a larger time overhead is expected, given that the horizontal coloring derives from failing

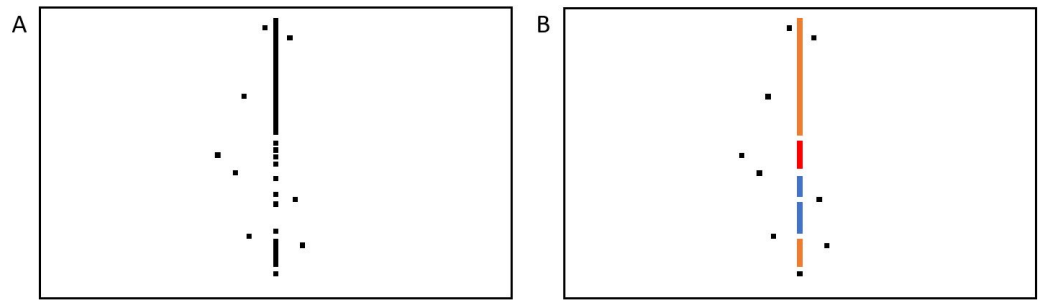


Figure 3.28: Example of partially failing bitline oriented scenario

masks that contains more than one failing bit, therefore taking more time to compute, and the PBIST is encountering faults at every consecutive read. For such a

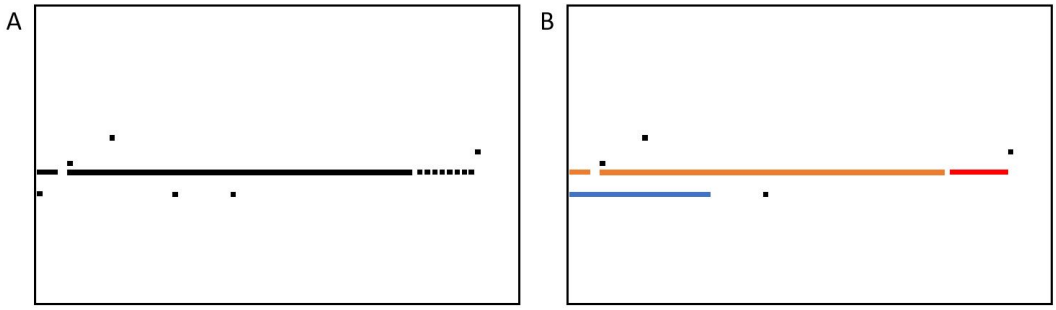


Figure 3.29: Example of partially failing word-lines

case, with 18,229 faults overall, the increase of test time with respect to the bit-by-bit approach is 69%, while the RAM memory save is about 98.68%. Table 3.4 shows a comparison of the bit-by-bit versus the proposed when varying the number of failing bits. It is worth noticing that our approach requires slightly more time and memory than the bit-by-bit approach for the sparseness of the faults that prevent their aggregation.

Sparse faults configurations are another important scenario to observe. This kind of constellation is the most difficult to deal with because of the limited compaction possibilities. Figure 3.30 depicts a quite dense case of sparse fails over the memory matrix. Some of them are originating a blue-colored slice when they are far but aligned on the same bit-line or word-line shape is a composite of the previously considered case. Despite the intrinsic difficulty, the proposed approach shows a limited loss with respect to the bit-by-bit approach. About the case in Figure 3.31, which contains 9,949 faults, the time to collect the diagnostic information is 680ms, while the bit-by-bit approach demands 440ms. Conversely, the memory occupation dramatically drops from 38.85KB for the bit-by-bit to 0.1KB for the proposed one.

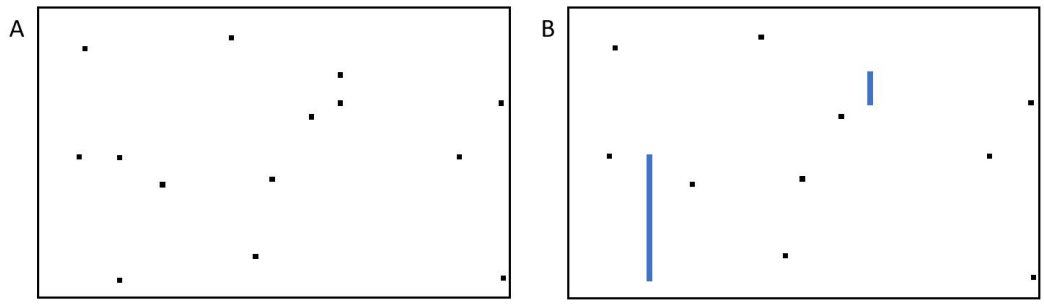


Figure 3.30: Example of a sparse failure scenario

Table 3.4: sparse faults scenario analysis

Method	# faults	10	100	1000
[2]	Memory occupation	40B	400B	3.9KB
Proposed		60B	600B	5.89KB
[2]	Test time	72ms	75ms	109ms
Proposed		74ms	78ms	139ms

Results on a larger base of devices

Experimental measurements are also available for a more extensive base of devices. We considered 1864 failing devices that come from front-end wafer testing operations [**b12**]. Such a set collects many different shapes and is used to further evaluate the benefits and costs of the proposed approach. We compare it with the bit-by-bit coordinate-based approach [2] and the compression approach in [16].

Table 3.5 reports about the capacity of the approaches in terms of number of devices over the entire base which can be bitmapped without exciding the on-chip 24KB RAM limitation. From this table it can be seen how many devices are fully logged within the 24KB only by our approach (135) and by the bit-by-bit coordinate-based approach.

When the 24KB of on-chip RAM available are filled, some diagnostic information are lost, unless the test and diagnosis process is interrupted, the current bitmap is dumped to the tester, and then the memory test procedure is resumed. Suppose the diagnostic collection is suspended and new faults are not logged anymore. In that case, the previously encoded faults are retained, and a partial failure constellation can be reconstructed at the end of the testing flow like the one shown in Figure 3.32. The selected population shows an average of around 2,000 faults, with a variance of around 5,000 faults. By looking at the performances over the population sample, we observed that the overall time to test and collect diagnostic information is 192ms on average, with a variance of 279ms. For the bit-by-bit approach, these values are 152ms on average with a 189ms of variance. Table 3.6

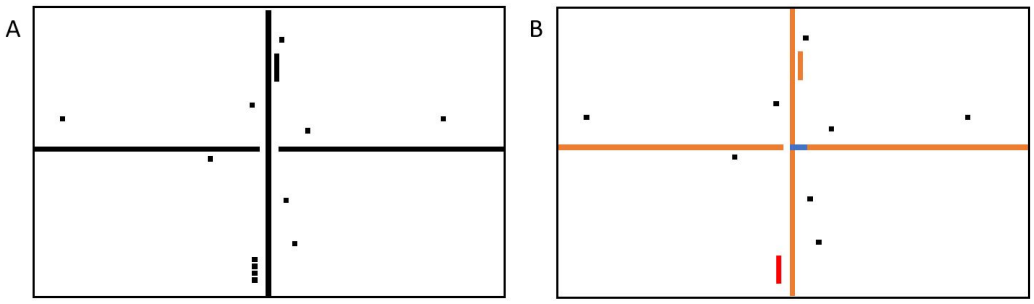


Figure 3.31: cross shaped failure constellation with working bits at the intersection

Table 3.5: confusion matrix comparing capabilities of proposed and reference approach [2]

		Proposed approach	
	Full bitmap in 24KB	Yes	No
[2]	Yes	89.32%	0.05%
	No	7.24%	3.38%

concerns bitmap size occupation; it shows the percentage of the investigated population whose bitmap creation was smaller in size, comparing our approach and the reference one. Considering all devices, the proposed approach asks for less memory in about 60% of cases. If failing scenarios with more than 250 faults are only considered, our approach shows size advantages in about 91% of cases.

Table 3.6: On-chip memory demends of proposed and reference [2]

	Proposed better than [2]	[2] better than proposed
All Devices	60%	40%
Devices with more than 250 faults	91%	9%

Concerning the comparison with [16], we computed the Pearson correlation index to measure the amount of differences with our approach. Constraining the memory requirements to 20KB, [16] can store any failure constellation with a pretty low accuracy; on average, we computed that the correlation index is 61%. Figure 3.33 shows the rebuilt failure constellations when a) it is compressed with [16] and b) when it is compacted with our compaction method. In this specific case, with around 2,000 faults, the correlation index is 83%.

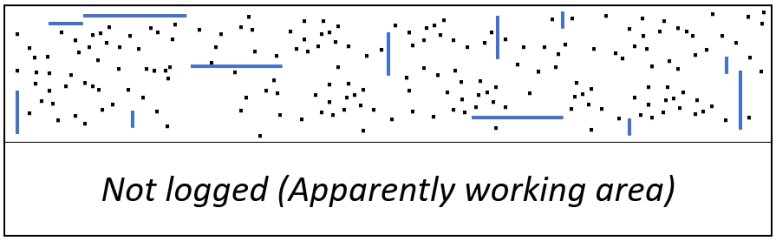


Figure 3.32: Partially reconstructed failure constellation from a full 24KB buffer

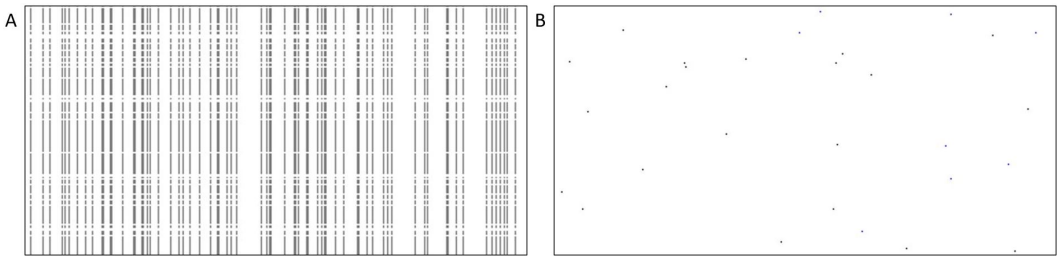


Figure 3.33: comparison among bitmap reconstructed by the method shown in [16](A) and the proposed compation one (B)

3.3.2 Delta-Oriented Compaction Algorithm

The proposed approach starts from the method disclosed in [24], expanding its capabilities to report just the differences between a base test and the subsequent tests. In this way, the absence of information serves as information itself, and only new (or disappearing) faults are encoded. The concept is graphically explained in Fig.3.34. In the example, a certain number of faults were discovered and encoded in the first test. Then, a second test is executed, resulting in a similar fault shape configuration. As can be seen from Fig. 3.34, the proposed algorithm just encodes the differences between the first test and the following one, reducing the amount of diagnostic data required while still maintaining the lossless nature of the method presented in [24].

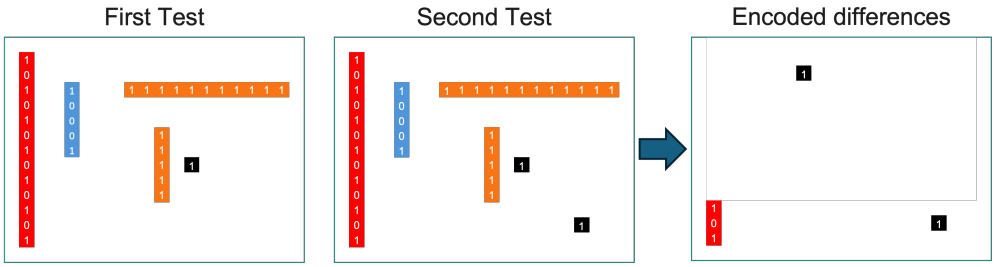


Figure 3.34: Encoded differences after two memory tests.

To achieve its goal, the algorithm efficiently stores the order of fault arrival in colored segments called delta slices, as can be seen in Fig. 3.35. These slices are temporary and only used internally by the algorithm to save faults' order of arrival. In the example, the memory is tested from left to right and from top to bottom. The resulting Delta Slices are placed in an ordered structure with the first slices containing the first fault found.

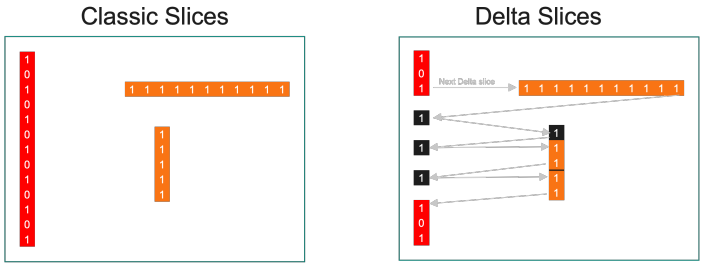


Figure 3.35: Classic slices created from [24] and the temporary Delta ones that represents the order of faults arrival.

From the second memory test onward, the algorithm computes the Next Expected Fault (NEF) and check it against the received faults. For example, at the beginning of the second memory test, the algorithm picks the first fault received during the first test and takes it as the first fault of the first Delta Slice. Every time the algorithm receives a fault to analyze, it follows Algorithm 2.

Algorithm 2: Fault Handling Algorithm

```
if incomingFault before NEF then
    encodeDifference(incomingFault);
else if incomingFault after NEF then
    while incomingFault after NEF do
        encodeDifference(NEF);
        NEF ← selectNextNEF();
```

Experimental Results

To validate the proposed approach, a simulator was developed in Rust to perform both [24] and the proposed algorithm. The simulator was fed with data coming from 10,000 devices modeled to realistically represent fault evolution between memories undergoing a typical Automotive eMemories test flow composed of four memory tests. A simplified example of how this devices are modeled is shown in Fig. 3.36. An algorithm generates realistic fault shapes for the first memory step and then expand them and/or adds other faults for the subsequent tests.

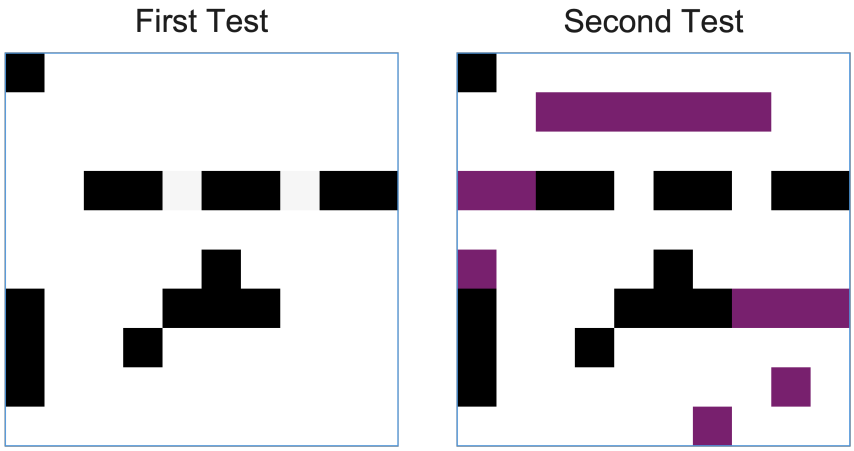


Figure 3.36: In the modeled devices at each subsequent test faults are expanded and/or new ones are created (purple squares).

The experimental results on these 10,000 devices are summarized in Table 3.7.

Table 3.7: Aggregated data for 10,000 devices tested with the methods in [2], [24] and the proposed approach.

For 10000 devices	Approach in [2]	Approach in [24]	Proposed approach
Memory Usage (MB)	1865.9	72	23.8
Memory Saving wrt to [24] (%)	-2591.5%	//	66.99%

As Table 3.7 shows, the proposed approach on average saves 66.99% of diagnostic memory space when compared with the method in [24], meaning that devices under test would be able to store more amount of data in case of limited on-chip diagnostic memory space.

3.3.3 Density-Oriented Compression Algorithm

The proposed solution is based on the compression of diagnostic failing information during the execution of an on-chip memory verification. Our solution can overcome the explosion in memory requirements required by [2] and, for sparse faults, by [24] to represent fault-dense scenarios.

The base element of the compression algorithm is called pixel, which represents a portion of the memory of the DUT. A pixel is composed of a configurable number of wordlines and bitlines. These two parameters are freely configurable to achieve

various levels of compression. The smaller the pixel's dimension, the greater the resolution at the expense of the memory requirements. Each pixel comprises a two-dimensional coordinate and a counter representing the number of faults for the represented memory portion.

Method	Density accurate	Storage	On-Chip
Landzberg[2]	Yes	High	Yes
Chen[16]	No	Variable with resolution	Possible
Bernardi[24]	Yes	Low	Yes
Proposed	Yes	Variable with resolution	Yes

Table 3.8: Comparison between features of various approaches

Table 3.8 shows a comparison between our proposed approach and other diagnostic data collection algorithms. Our proposed approach is topologically accurate, showing the density of memory fails in different areas of the memory. This characteristic is in common with the algorithms shown in Bernardi et al. [24] and Landberg [2]. As for storage requirements, similar to the algorithm of Chen et al. [16], it varies based on the chosen resolution. In common with Bernardi et al. and Chen, the proposed approach's memory requirements are lower than the Landzberg approach. Lastly, all the compared methods are executable on-chip (with [16] being originally tested with additional hardware and tester capabilities).

The proposed approach does not require additional hardware in the DUT. It is also compatible with BIST-based memory tests, as well as the CPU-based ones. Moreover, the proposed approach can be used in conjunction with the methods [2] and [24] by dynamically switching to the compression approach when the available dedicated on-chip storage is running low.

SLAC Pixel structures

Each pixel is encoded into the on-chip memory as a structure that includes three parameters: x and y coordinates and the number of faults. This approach allows the algorithm to achieve the minimum possible used space by exploiting bit-wise operators to manage the structure parameters. Each pixel is represented using 4 Bytes in the following way (Fig. 3.37):

- 1 Byte, X pixel coordinate
- 1 Byte, Y pixel coordinate
- 2 Bytes, faults counter

Pixel	
X	1 Byte
Y	1 Byte
Faults	2 Bytes
TOTAL	4 Bytes

Figure 3.37: Pixel encoding structure

On-chip memorization of the encoded information

When a new fault is discovered and processed by the proposed algorithm for compression:

- The corresponding pixel X and Y coordinates are identified. These two values are computed in the following way:

$$X_{\text{Pixel}} = *\frac{X_{\text{Fault}}}{bitsPerPixel} \tag{3.8}$$

$$Y_{\text{Pixel}} = *\frac{Y_{\text{Fault}}}{bitsPerPixel} \tag{3.9}$$

  Where $bitsPerPixel$ corresponds to the pixel's dimension terms of number of bitlines or wordlines. X_{Fault} and Y_{Fault} represent the fault coordinate.

- The pixel with the corresponding X and Y coordinates is indexed, and the counter used to represent the number of faults is increased by one.

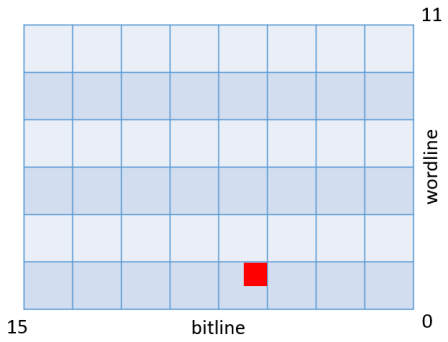
To optimize the algorithm's speed, available memory has been organized using a set-associative approach as in Bernardi et al. [24]. Given a number of N sets, the allocated on-chip memory for the diagnostic data is equally divided into N portions. When a new fault is found, the corresponding pixel is indexed using the X coordinate. The pixel location is used to generate the set and tag in the following way:

$$SET = X_{\text{Pixel}} \% N \tag{3.10}$$

$$TAG = \frac{X_{\text{Pixel}}}{N} \tag{3.11}$$

Each SET represents a different list that is indexed once a new fault is found. A linear search is then performed over all pixels already present in the selected SET, checking for correspondence using the TAG.

For example, Fig. 3.38 depicts a memory composed of 16 bitlines and 12 wordlines. In this example, each pixel represented by a square composed of 2 bitlines/-wordlines each ($bitsPerPixel = 2$). The red square represents a fault located at coordinates $X = 6$ and $Y = 1$. Fig. 3.38 shows an example of how the SET and TAG of the pixel are computed starting from a given fault.



$X = 6$ $\quad X_{\text{Pixel}} = 3$

$Y = 1$ $\quad Y_{\text{Pixel}} = 0$

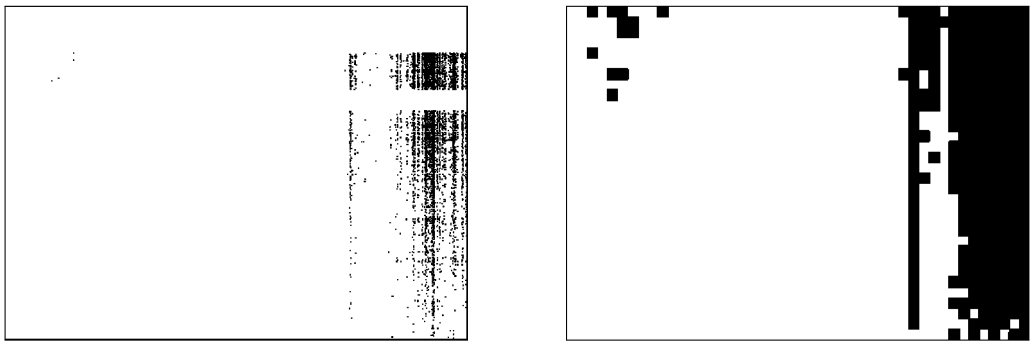
$SET = 1$

$TAG = 0$

Figure 3.38: Example of set and tag computation starting from a fault

Another important aspect is that a pixel is stored in the on-chip memory only if there is at least one fault. The first fault will trigger its creation. This approach can reduce the storage used to the minimum, avoiding storing useless information.

Fig. 3.39a represents a Bitmap on a memory bank using a compaction algorithm [24]. This algorithm is based on encoding and compacting faults while performing on-chip memory verification. To reduce the used storage, faults are compacted and encoded as contiguous slices. In this case, only a portion of the memory could be represented due to the on-chip storage limitation. Fig. 3.39b represents the same failing information using our compression algorithm. The compressed bitmap allows to overcome the storage limitation. Each square corresponds to 128 wordlines and 128 bitlines and has been colored if the portion has at least one fault.



(a) Bernardi et al. (b) Proposed

Figure 3.39: Bitmap on one single bank with sparse faults, presenting a topological failings pattern towards the right

Fig. 3.40 shows a zoomed version of the example in Fig. 3.39 reconstructed starting from the data produced by the algorithm of Chen et al. [16]. The resolution was set to the maximum allowed to represent the entire memory in the dedicated on-chip memory reserved for diagnostic information. The red dots in the picture represent faults, while the blue dots represent the area in which the presence of faults is uncertain. For this uncertainty, Chen et al. method is unsuitable for characterization studies, where the topological fault distribution is the most crucial parameter for designers and technology experts.

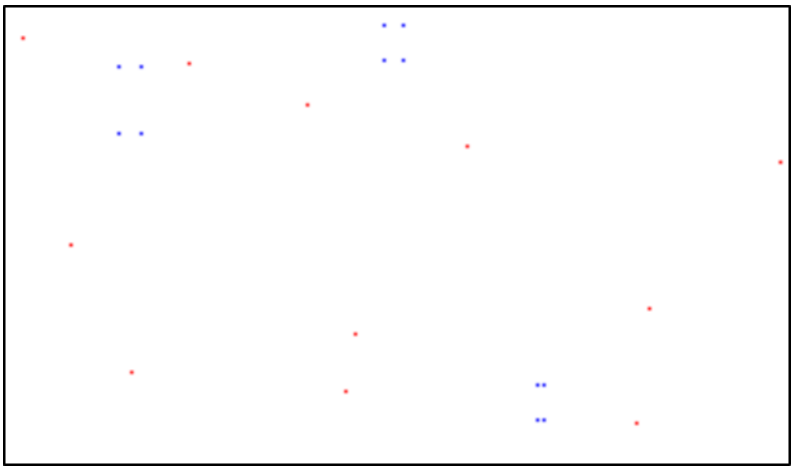


Figure 3.40: Example of reconstructed memory using Chen et al. algorithm

Color gradient for visualization

To better visualize the generated pixels, we developed a post-verification tool that creates a colored representation of the memory under test. This tool runs offline on the tester or on an external computer after collecting the pixels from the DUT. The corresponding color depends on the number of faults represented by the pixel. A naive approach would be to divide multiple ranges of faults and assign each to a different color. The problem with this solution is that the entire spectrum must be manually assigned. To overcome this limitation, the value is mapped into the three RGB components. Each pixel that has at least one fault is color mapped using three different sine functions, centered respectively at 0.0, 0.5, and 1.0 in the following way:

$$\begin{aligned} \text{RED} \quad & sin(\pi \cdot faults_{norm} - \frac{\pi}{2.0}) \\ \text{GREEN} \quad & sin(\pi \cdot faults_{norm}) \\ \text{BLUE} \quad & sin(\pi \cdot faults_{norm} + \frac{\pi}{2.0}) \end{aligned} \tag{3.12}$$

Where $faults_{norm}$ refers to the normalized number of faults with respect to the maximum number of representable ones by a pixel (on turn dependent by its dimension):

$$faults_{norm} = \frac{faults_{count} - 1}{faults_{max}} \tag{3.13}$$

Value of $faults_{max}$ can be offline trimmed to highlight better topological patterns that appear with higher or lower faults.

The plot of the three sine functions is available in Fig. 3.41. Coloring examples are depicted at Fig. 3.42.

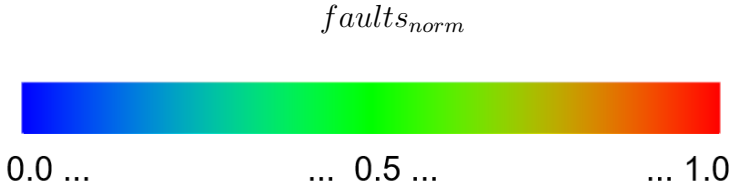


Figure 3.41: Mapping function from pixel difference to RGB components

The post-processing function is applied offline once the failing information have been downloaded from the test machine. A different color schema can be applied depending on the needs, making this approach even more flexible. In the shown case, the higher the fault density, the more red the pixel will be. A lower number of faults will make the color bluer.

Experimental Results

This section shows our experimental results collected with the proposed compression algorithm using an Automotive SoC device made by Infineon.

The following results are collected while characterizing the eMemories sense amplifier's reference current. In our experiments, we had a limitation of an embedded 24KB of memory to collect diagnostic information from these tests. All the results and pictures that will be shown are contained in this integrated memory. Once this memory is saturated by diagnostic information, the data logging stops leading to a loss of information. The following results show the performance of the various bit mapping algorithm with this memory limitation.

Data visualization

The following pictures illustrate how the same fault configurations are exported and visualized using different approaches. The main goal of our approach is to help designers and technology experts visualize the failure density of their DUT, so the

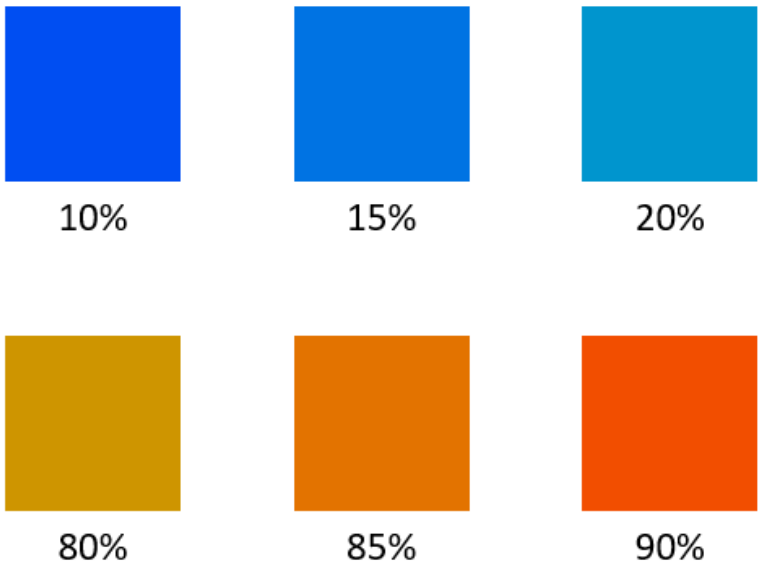
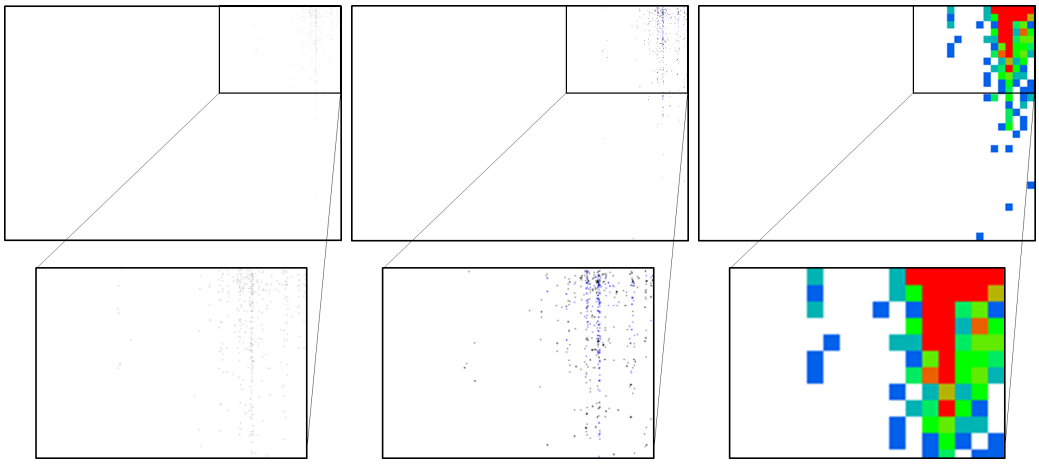


Figure 3.42: Example of pixel coloring based on the percentage of faults

main focus of this paragraph is to show how this density can be discerned using different methods.

The following pictures have been obtained by gradually shifting the reference current of the sense amplifiers, a standard test performed during the characterization of new devices.

Fig. 3.43 shows the first of these current reference shifts performed on a single bank of the memory under test. In this picture, there is a comparison of three different diagnostic data collection algorithms. In Fig. 3.43a the simple Landzber bitmapping approach [2], in Fig. 3.43b the compaction algorithm of Bernardi et al. [24] and in Fig. 3.43c the proposed compression approach. As can be seen, by using A, identifying the most fault-dense area is not immediate. Even a zoomed-out version, centered on the most fault-dense zone, is challenging to analyze and observe. This situation is slightly better for the compaction algorithm in 3.43b, where the colors help identify the faults. Using 3.43c, the topological distribution of faults is immediately evident, indicating the most fault-dense zones on the top right corner of the memory bank. In the following, we will keep the Bernardi approach as a reference for a lossless algorithm.

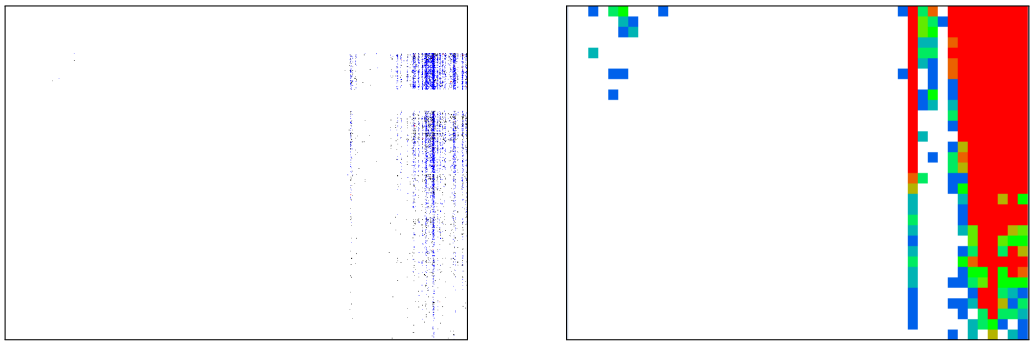


(a) Landzberg (b) Bernardi et al. (c) Proposed

Figure 3.43: Bit fail map visual comparison with slight reference current shifting (increment #1)

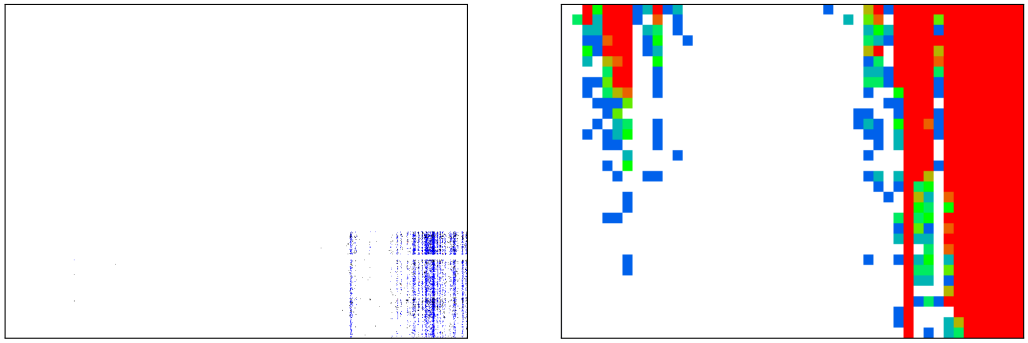
Fig. 3.44 further increases the shift in reference current of the sense amplifiers. The fault density is again more visible in Fig. 3.44b, showing the reconstruction of the bank with the proposed approach. The algorithm of Bernardi et al. in Fig. 3.44a tries to show the same information in a lossless compaction of the data but is not able to represent the faults in the top left corner of the bank that are starting to fail under the tighter reference current constraints. The Bernardi et al. approach saturates the 24KB of available memory to represent the fault information, reducing the amount of helpful information that reaches the ATE at the end of the test.

Fig. 3.45 and figure 3.46 clearly show that the lossless approach cannot give helpful information about the fault happening in the memory under test. The 24KB limit is reached when just a portion of the total faults are discovered, and a vast percentage of memory is not represented at all. On the other hand, the proposed algorithm can correctly represent all the faults in the memory, albeit in a compressed manner.



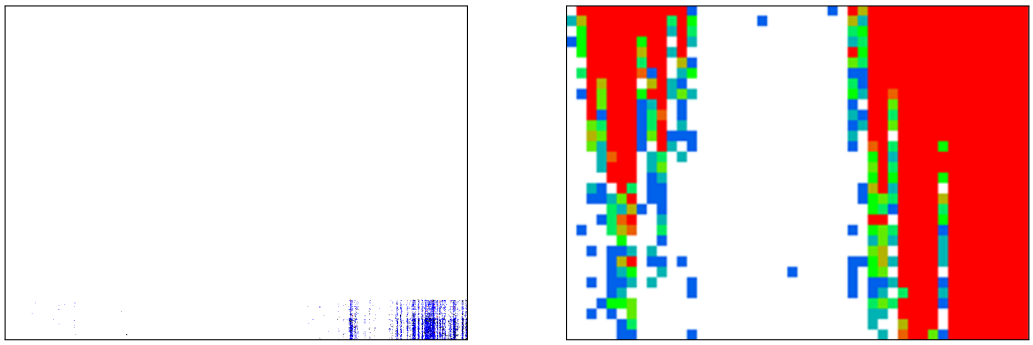
(a) Bernardi et al. (b) Proposed

Figure 3.44: Bit fail map visual comparison reference current shift (increment #2)



(a) Bernardi et al. (b) Proposed

Figure 3.45: Bit fail map visual comparison reference current shift (increment #3)



(a) Bernardi et al. (b) Proposed

Figure 3.46: Bit fail map visual comparison reference current shift (increment #4)

TIMING

In this section, the timing overhead of the various algorithm is analyzed. The Landzberg approach is always the fastest and will be used as the overhead computation reference. The scenarios considered to make this consideration have a variable amount of faults randomly distributed along the memory under test. As shown in Table 3.9, the overhead of Bernardi et al. and the proposed approach grows with the number of faults found. The overhead of the proposed approach is always lower than the Bernardi et al. one, especially with the growing number of faults. For example, at 5500 faults, the Bernardi approach has a 14.63% of overhead, while the proposed stops at 9.60%.

	Time overhead (% wrt to Landzberg)	
# Faults	Bernardi et al.	Proposed
100	2.33	2.28
250	2.67	2.48
500	3.02	2.67
1000	4.18	3.25
2000	6.81	4.48
4000	11.03	7.35
5500	14.63	9.60

Table 3.9: Comparison between timings of various approaches in a memory with randomly distributed faults (with % computed over the overhead with respect to the Landzberg approach)

Memory requirements

This section is focused on the memory needed to represent the various scenarios depicted in figures from 3.43 to 3.46. These cases are taken from a real characterization step performed by Automotive SoC manufacturers. The diagnostic information collection limit is 24KB of on-chip dedicated memory. When this memory is saturated, no additional data is saved, and the related information is lost. It would be possible to notify the ATE of the saturation, wait for it to download the 24KB, and restart the logging. However, this communication with the ATE would be unmanageable for fault-dense scenarios and cases of multiple devices tested in parallel, such as in a mass production environment. As seen from Table 3.10, the case in Fig. 3.43 is the only one in which all three analyzed methods can fit the diagnostic data in 24KB of dedicated on-chip storage. The Landzberg approach is already near this value, with a memory requirement of 20.62KB. The Bernardi et al. approach requires 90% less with requirements of 2.19KB, a reduction of 90% with respect to Landzberg. Finally, our proposed approach required 0.60KB, a reduction of 97%

with respect to Landzberg and of 72,6% with Bernardi et al. In the other cases, up to 3.46, the lossless algorithms saturates the 24KB available, with the proposed approach requiring only 4.68KB in the worst case. For a visual comparison, Fig. 3.46 clearly shows the limitation of the lossless approaches. With 24KB, only a tiny portion of the memory is represented. In this case, our compression approach can represent all the memory under test, with information about the fault density in its various zone, invaluable information for Automotive SoC manufacturers.

	Size (KB)		
	Landzberg	Bernardi et al.	Proposed
Fig. 3.43	20.62	2.19	0.60
Fig. 3.44	**24.0**	**24.0**	1.82
Fig. 3.45	**24.0**	**24.0**	2.04
Fig. 3.46	**24.0**	**24.0**	4.68

Table 3.10: Comparison between memory requirements of various approaches

3.4 MBIST Architecture and Management for Parallel Testing

The increasing complexity of modern System-on-Chip (SoC) designs has led to a significant rise in the number of embedded memories, making efficient testing and diagnosis of these memories a critical challenge. Memory Built-In Self-Test (MBIST) architectures have become a standard solution for ensuring the reliability of embedded memories. However, as the number of memory banks grows, traditional testing methods face limitations in terms of test time, area overhead, and diagnostic efficiency. Addressing these challenges requires innovative approaches that balance performance, resource utilization, and diagnostic accuracy.

This section presents a novel MBIST architecture and management scheme designed to enable parallel testing of memory banks while minimizing test time and hardware overhead. The proposed approach leverages a combination of CPU firmware and a lightweight hardware add-on to efficiently manage MBIST operations. By introducing a common status register and employing advanced search algorithms, such as binary search and binary search with priority shifting, the architecture achieves significant improvements in test time and diagnostic fairness compared to traditional linear search methods. This work has been published in the MDPI Electronic Journal[42].

3.4.1 Proposed approach

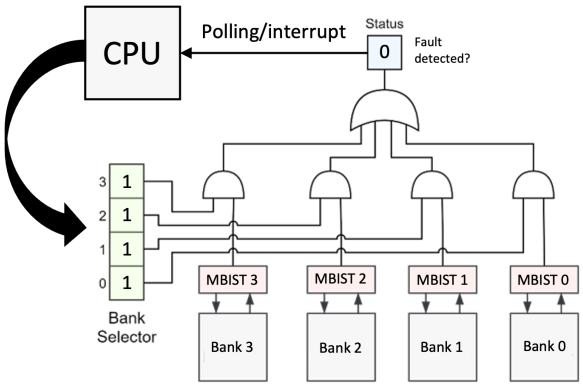
The proposed approach considers the usage of CPU firmware supported by an additional hardware structure, as illustrated in Figure 3.47A). Such ad hoc circuitry controls a status bit value by ORing the status bit spilled from every MBIST. When a fault occurs in any memory, this flag bit raises to value one and triggers the CPU similar to an external interrupt. The processor, left in idle mode, wakes up from idle and interacts with a bank selector register to identify the failing bank following the approach shown in the following paragraphs. This hardware organization is cheap because of CPU reuse to support memory testing and because it demands very few extra gates. For sure, a single flip-flop could be extended to a register. However, it would not fit other constraints, such as the number of interrupt channels, other than for area overhead minimization. A status register without interrupt capabilities would require continuous polling by an always-awake CPU, which would impact device consumption during the test execution.

Therefore, distributed MBISTs supported by the proposed hardware add-on allow the execution of concurrent memory bank tests. Consequently, it sensibly reduces test time with small area and power demands. This approach focuses on getting maximum performances out of this scheme without sacrificing the effectiveness of the collected failure coordinates, especially in cases of simultaneous fails detected on different MBISTs.

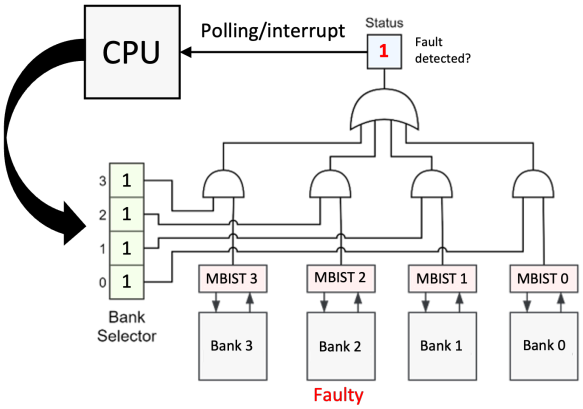
Overall, we compare three different methods:

- A linear search method, which is considered as a baseline for comparison.
- Two versions of a binary search method: without and with priority shifting.

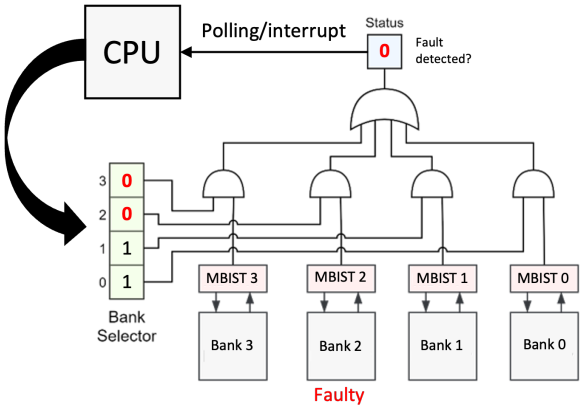
Among these methods, binary search with priority shifting achieves a better trade-off between test time and the retrieved failure information quality, i.e., overcoming the limits highlighted in Section 2.1.5 about diagnostic data collection.



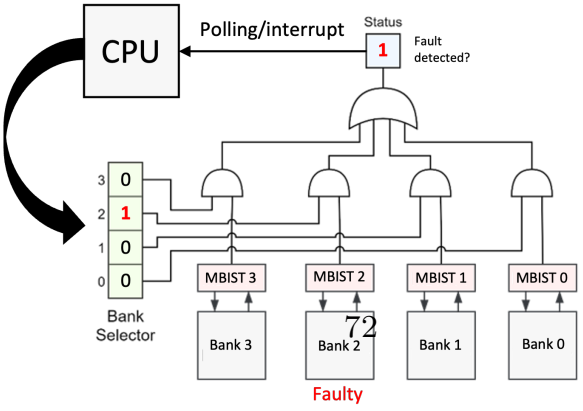
(A) First step: all MBISTs selected and no faults detected



(B) Second step: all MBISTs selected and at least a fault detected



(C) Third step: two MBISTs selected and at least a fault detected



(D) Fourth step: a single MBISTs selected and a fault detected

Figure 3.47: Example of a real scenario.

Hardware Architecture for Diagnosis

The hardware design used for our approach enables concurrent access to one or more MBIST status registers by providing the ORed version of all the selected ones in one readout. Considering the case in which all MBISTs are addressed for a status register readout, a raised stop-on-fail flag means that at least one module needs to be served by the CPU.

Figure 3.47A depicts a simplified model of the proposed hardware organization. The status register bit accessible for readout contains the ORed information of all MBISTs selected through the bank selector. That is an array composed of as many bits as there are MBISTs available. The output of an MBIST is enabled when its corresponding bit in the bank selector is set to one. To understand if at least one MBIST found a fault, the CPU must check the common status register bit through polling or by waiting for an interrupt. Independent of the chosen solution, this organization shows clear advantages in circuit complexity and test time compared with an arrangement wherein each MBIST has its own status register. Reducing the number of needed registers is a major contributor to area savings. For what concerns the test time, as we will see in the following section, this organization allows for clever search algorithms that exploit the common status register bit. Architectures with completely independent MBISTs, such as the one presented in [5], can only resort to the linear search algorithm, which will be used as a reference to evaluate the performance of the proposed algorithms.

Binary Search Approaches

The goal of the proposed search algorithm is to overcome the intrinsic problems of linear access that affect the overall test time. Taking advantage of the capability of our proposed schema, it is possible to shorten the test time by relying on a tailored binary search procedure such as the one described in Algorithm 3.

Algorithm 3: Binary search for N MBISTs

```
Start all MBISTs;
Start ← 0;
End ← N − 1;
Enable all MBIST outputs;
while Test not finished do
    if common status bit is 1 then
        while end − start ≠ 0 do
            middle ← (start + end)/2;
            Enable output for MBIST[start] to MBIST[middle];
            if common status bit is 1 then
                end ← middle;
                Continue;
            else
                start ← middle;
```

In our scenario, the binary search algorithm generates the bitmasks to be loaded into the bank selector.

Figure 3.47 shows how binary search is applied in the case of four MBISTs by disabling or enabling the outputs of groups of MBISTs. Once a fault has been found by at least one MBIST, the ORed stop-on-fail flag is raised. This flag generates an interrupt that wakes the CPU from idle. At this point, the algorithm generates a bitmask containing "half 0s" and "half 1s": enabling the ORed flag of MBISTs 0 and 1 (half the MBISTs available). The algorithm will continue analyzing the rightmost half if the flag is high. Otherwise, it will proceed on the leftmost half. The analysis iterates, halving the number of MBISTs to evaluate step-by-step. At the last step, a bitmask containing only one bit set to "1" is applied, and the procedure returns the index of the bank to be served.

With this approach, in case of fault, exactly $\log_2(N)$ checks are needed to identify the correct MBIST. In case of no faults, one check is enough for all the MBISTs. Anyway, the pure binary search approach still suffers the intrinsic propriety order of access seen in Section 2.1.5. For example, let us analyze the case in which every bank requests an action from the CPU (Figure 3.48a): the selected bank to be served will inevitably be the rightmost one. This unintended fixed priority could lead to undesirable starvation of "stopped" MBIST modules. Consider, for example, an event wherein one memory module completely fails and the rest present only one fail on the first page; suppose that the entirely failed memory module has the highest intrinsic priority. All the MBISTs will start concurrently, and they will all stop on the first page. The CPU will serve and repeatedly resume the entirely failed bank, preventing the other MBISTs from stepping over the first failing page

and proceeding concurrently.

To overcome this problem, we developed a revised binary search by merging the superior performance of the binary search with the starvation-free capability of round-robin scheduling.

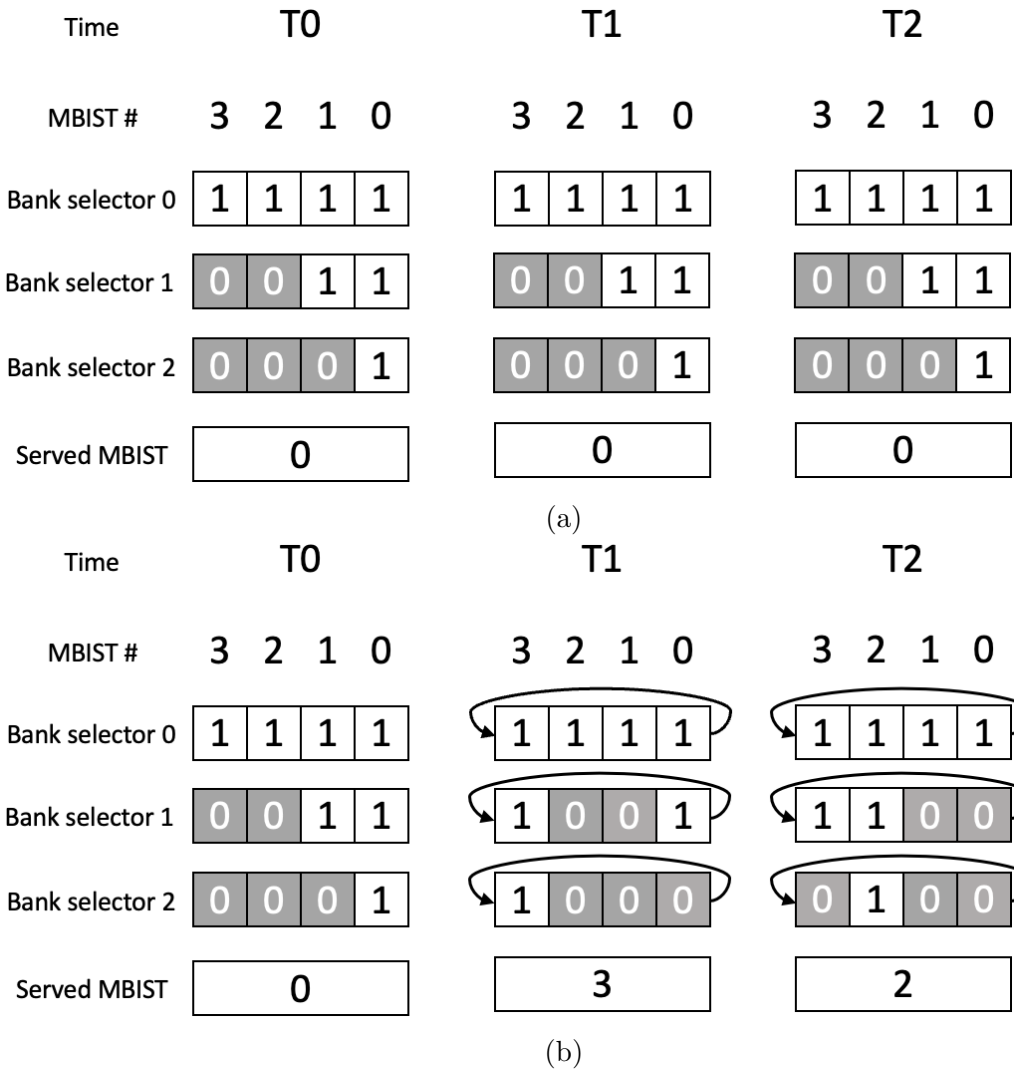


Figure 3.48: (**a**) Binary search, fixed priority. (**b**) Advanced binary search: the bank selector rotation increases at every iteration.

To achieve this priority changeover in the served MBIST, an n-bit rotation to the bank selector generated at every step of the algorithm is applied: n is increased by one each time "binary search with priority shifting" is executed.

Figure 3.48b analyzes again the case in which multiple MBISTs need to be served by the CPU at the same time: applying a 1-bit right rotation to the bank selector,

the priority is shifted from the rightmost module (MBIST0) to the leftmost one (MBIST3).

For manufacturers, it is crucial to have as much information as possible about the failures in their systems. However, in a production environment, a trade-off between the accuracy of the collected information and its costs must be explored; this includes test time and diagnostic memory size.

A common approach when there are memory limits consists of assigning a fixed amount of on-chip memory for storing as much diagnostic information as possible. This constraint limits the number of fault locations that can be stored on-chip during the tests. Without a specific division of the available on-chip memory between the different MBISTs, a manufacturer may lose important insight into of some the memory units under test. For example, in a scenario with a memory bank showing a lot of failures (enough to fill the assigned memory space) and another with a few, if the priority is fixedly given to the MBIST self-testing the memory with more failures, then the fault information from the other one may not be collected. For a producer, this is the worst possible situation; it would be much better to renounce some information about largely failing banks than totally missing indications about a failing memory bank.

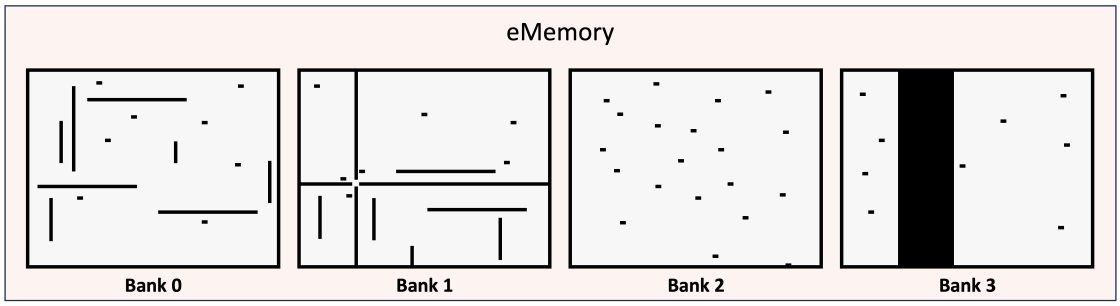
Figure 3.49a–c illustrate the issue. It uses a memory module with four banks as a reference; each bank has its own MBIST.

A Figure 3.49a shows all the failures in the memory. Without an on-chip memorization limit, the total test time is the only difference between the three access methods presented in this thesis.

B In Figure 3.49b, there is a limit to the on-chip memory available to store diagnostic information. Using the linear and binary search approaches, the BIST of Bank 0 is always served first. Bank 1's BIST is served next and fills the remaining reserved memory space. As can be seen, no information about Banks 2 and 3 reaches the external world as the reserved memory is already saturated when their MBISTs are served.

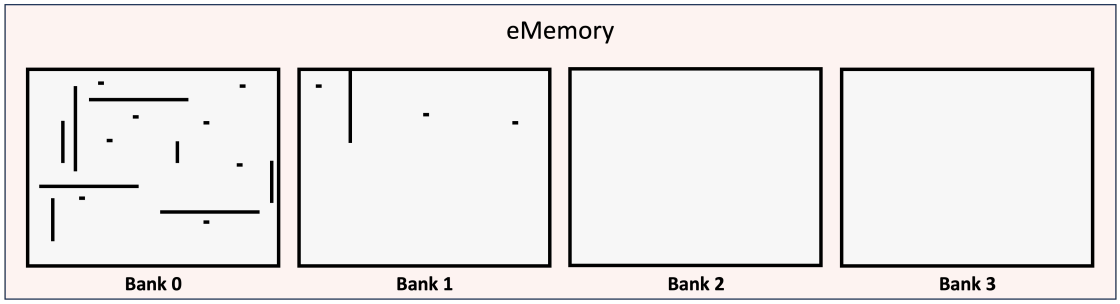
C Figure 3.49c shows the effect of the binary search with priority shifting in case the memory limitation is still in place. The result is a fairer memory representation with partial information from all the memory banks.

Binary search with priority shifting ensures a fair representation of the different banks in the memory. Of course, the trade-off is represented by the partial representation of these banks. Nevertheless, it is essential to notice that it is often possible to extrapolate fault patterns and understand their causes even with partial information.

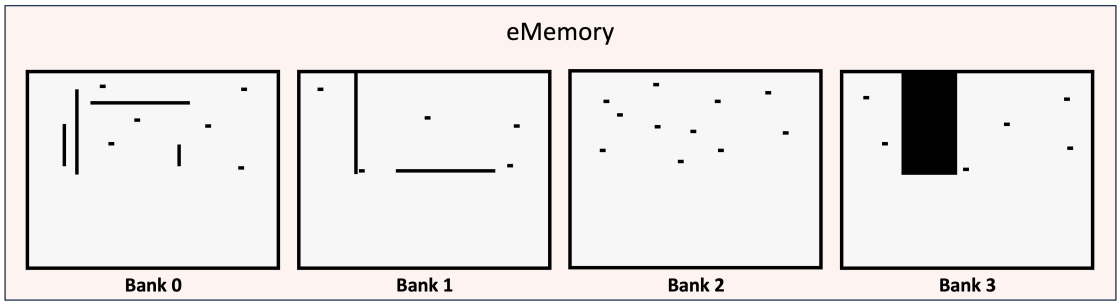


(a)

Figure 3.49: *Cont.*



(b)



(c)

Figure 3.49: (**a**) Unlimited diagnostic storage space. (**b**) Limited diagnostic storage space: binary search and linear search. (**c**) Limited diagnostic storage space: advanced binary search.

Experimental Results

This section analyzes the behavior of the proposed hardware and software scheme in real-case scenarios. We used a subset of the tests usually executed during the characterization of the SoC. As execution time is the best metric for this kind of test, we analyze the three access mechanisms' timing behavior. All of them

were executed in an Infineon™ SoC with the MBIST organization described by our approach and using polling to check the MBISTs' common status register. The following experimental results examine three scenarios: a sweep of artificially injected errors, data from real faulty devices, and some special cases. In the following sections, the linear search algorithm is used as a reference for comparison with the basic and priority shifting binary search algorithms. The two binary search methods exploit the architecture proposed in this thesis. Such optimized algorithms cannot be implemented in architectures with isolated MBISTs like the one presented in [5]; these architectures can only implement the linear search method.

Sparse Faults Randomly Distributed

Sparse faults randomly distributed along the memory are one of the common issues with embedded memory. To emulate the sparseness of defective bits that characterizes real devices, 5000 tests were performed under these conditions:

- Single-bit error (SBE) only: at most, a single bit per page can be faulty;
- Pseudo-randomly generated addresses: the indexes of faulty pages are calculated in a pseudo-random manner;
- Number of injected errors is between 0 and 90 k, which roughly corresponds to 0.03% of the bits in the memory.

The faults generated by this random approach are very similar to those obtained during tests performed during the characterization of the sense amplifiers of the DUTs and generate a semi-random distribution of faults along the entire memory.

Test Time Comparison

In our experiments, it is clear that the test times are directly proportional to the number of injected faults for all three analyzed algorithms in a linear manner. The superiority of the binary-search-based approaches results in a time reduction of roughly 25% compared to the linear search. The standard implementation provides the best performances, leading to a 26.3% test time reduction, while the binary search with priority shifting is limited to 23.9% (the 2.4% overhead is due to the rotation of the bank selector).

Table 3.11 shows our results with respect to the time overhead of the three methods with increasing fail counts. In this experiment, we injected variable amounts of sparse fails on a single device. As we can see, the binary search approaches have a sensible advantage over the linear one.

Table 3.11: Test time comparison for different approaches.

Sparse Random Fault Timing Comparison				
Fault Count (#)	27,451	41,416	54,959	68,817
Linear Search (ms)	92	138	182	228
Binary Search (ms)	68	102	134	168
Binary Search priority shifting (ms)	69	104	138	173

Progress of Bank Verification

To test the evolution of bank verification, we devised an experiment in which we injected pseudo-random SBEs into roughly 0.006% of the memory. This experiment aimed to prove that the linear search and binary search methods prioritize the banks with the lower index. In contrast, the binary search with priority shifting makes all MBISTs proceed in parallel to the greatest extent possible.

The line plots in Figure 3.50a–c (with the banks' color legend in Figure 3.50d) highlight how the intrinsic priority affects the progress of the MBISTs' memory verification. The x-axis represents the test duration, with 100% being the overall time that the linear search needs. The y-axis measures the ratio of faults found by each MBIST and the total number of errors in the corresponding bank, which is expressed as a percentage. The time reduction of the two binary-search-inspired approaches is over 25%.

This behavior can be easily discerned in Tables 3.12 and 3.13. These tables shows the percentages of faults analyzed by the three access methods in five different banks at 10% and 30% of the test time (with 100% representing the time required for the linear search).

For the 10% test time represented in Table 3.12, it can easily be seen that the linear search and the binary search have similar behavior:

- Faults in Banks 0 and 4 start to be analyzed, with Bank 0 receiving the most attention;
- All faults in the other banks are not analyzed.

By contrast, the binary search with priority shifting shows a "fairer" distribution of the analyzed faults. Bank 0 is still the most analyzed (perhaps because MBIST 0 is the first that the CPU programs and starts). Banks 4, 8, 12, and 14 are all analyzed to a certain degree based on factors such as fault location and, consequently, the detection time for all MBISTs.

For the 30% test time represented in Table 3.13, again, the priority of lower-indexed banks is clear in linear and binary searches:

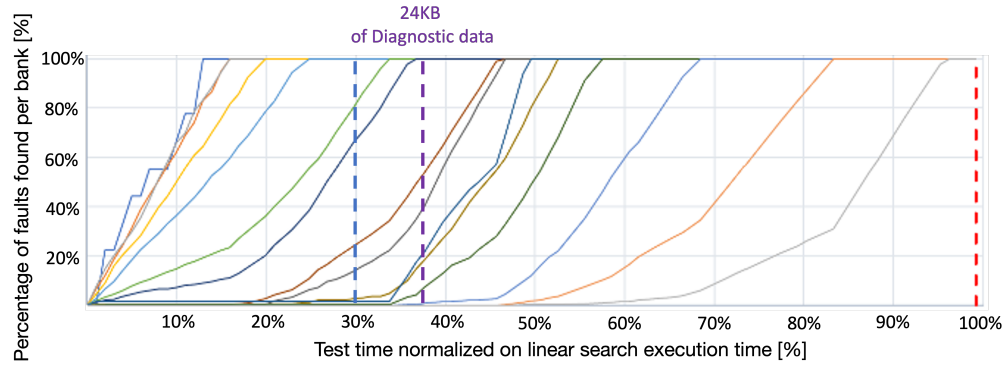
- Faults in Banks 0 and 4 are completely analyzed at 30% test time;

- Faults in Bank 8 are starting to be analyzed;
- Banks 12 and 14 are not even reached at this point.

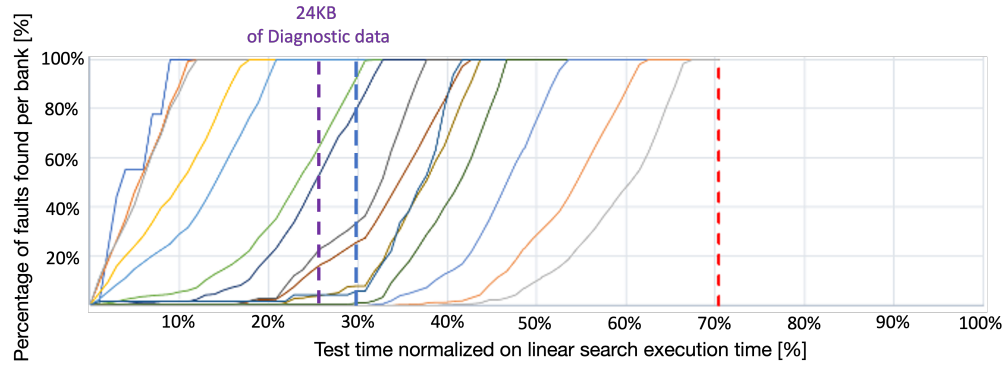
With the binary search with priority shifting, Bank 0 is still the most analyzed, while Banks 4, 8, 12, and 14 are all analyzed up to a certain degree.

Figure 3.50a–c also show what happens in case of a 24 KB limit for the representation of diagnostic information. The dashed purple lines represent the point at which the 24 KB are saturated in the different scenarios. The saturation happens in different conditions for the three tested algorithms:

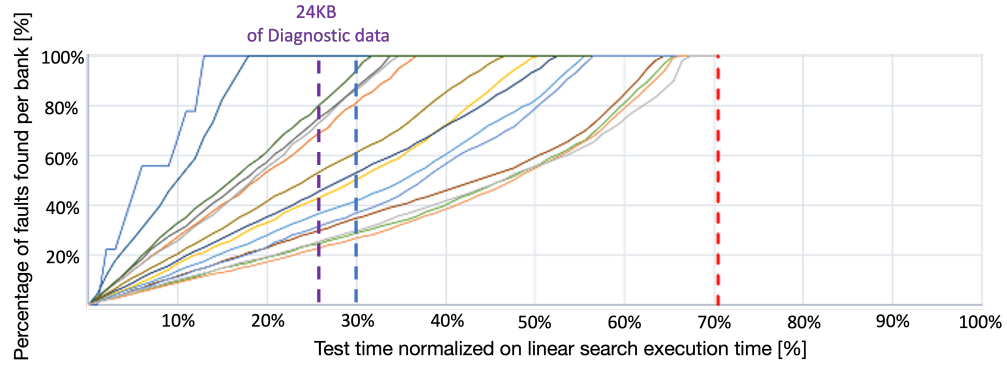
- For the linear and binary search algorithms, the intrinsic priority of the lowest-indexed banks is clear. Banks 0 to 4 are completely represented in both cases. The other banks are analyzed just partially or not analyzed at all, e.g., Banks 13 and 14.
- The saturation scenario is entirely different for the binary search with priority shifting algorithm. All the banks are at least partially analyzed, and a couple are already wholly analyzed. This scenario is much fairer in terms of representation. Once analyzed, the 24 KB of collected diagnostic information will reveal the partial situation of all the memory banks, making diagnosis of the device easier.



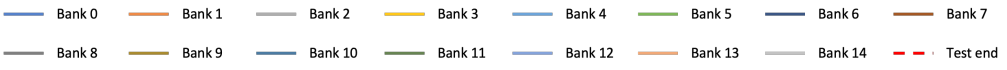
(a)



(b)



(c)



(d)

Figure 3.50: (**a**) Bank-by-bank test percentages over time using the linear search algorithm. (**b**) Bank-by-bank test percentages over time using the binary search algorithm. (**c**) Bank-by-bank test percentage over time using the binary search with priority shifting algorithm. (**d**) Color legend for the various banks.

These results are also summarized in Table 3.14, which shows the percentages of faults analyzed at the 24 KB diagnostic memory saturation point by the three access mechanisms for five different banks.

Table 3.12: Bank analysis at 10% test time.

Percentage of Stored Faults at 10% Test Time for Different Approaches					
Bank Number	0	4	8	12	14
Linear Search	66.89%	35.81%	0%	0%	0%
Binary Search	100%	28.68%	0%	0%	0%
Binary Search, Priority Shifting	67.97%	14.18%	30.32%	11.98%	10.51%

Table 3.13: Bank analysis at 30% test time for different approaches.

Percentage of Stored Faults at 30% Test Time					
Bank Number	0	4	8	12	14
Linear Search	100%	100%	14.4%	0%	0%
Binary Search	100%	100%	34.5%	0%	0%
Binary Search, Priority Shifting	100%	41.71%	87.5%	37%	29.9%

Table 3.14: Analyzed banks at 24 KB diagnostic space saturation for the different approaches.

Percentage of Stored Faults at 24 KB Diagnostic Memory Saturation					
Bank Number	0	4	8	12	14
Linear Search	100%	100%	38.03%	0.67%	0%
Binary Search	100%	100%	22.32%	0%	0%
Binary Search, Priority Shifting	100%	36.31%	75%	31.75%	25.28%

Data from Production Tests

Test times were evaluated by running the three algorithms on datasets of faults extracted from over 500 samples coming from devices in the production phase. These experiments were designed to prove the validity of our scheme in a real case scenario with real devices and its feasibility in an industrial scenario. This time, with respect to the pseudo-randomly generated faults, pages can also contain multiple-bit errors and other typical embedded memory fault shapes. Figure 3.51 illustrates the results for this case. The axes represent the fault collection time and the total number of faults.

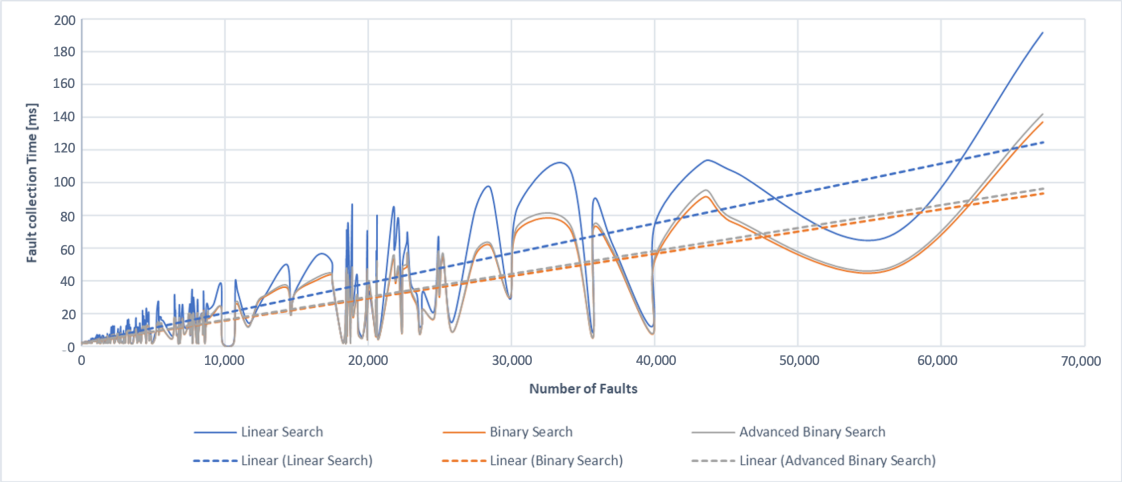


Figure 3.51: Test times against fault percentage in real case scenarios (solid lines) and random faults (dotted lines).

The test times do not increase linearly with the number of faults: pages are evaluated by the MBIST in one read operation, so multiple faults are analyzed at once. The advantage of the binary approaches is unequivocal when compared to the linear search and can be highlighted by the dashed trend lines of the three algorithms. Again, the proposed binary search approaches show a percentage reduction in the test times of roughly 25%. Once more, the binary search with priority shifting suffers a 2% overhead compared to the standard binary approach.

3.5 Machine learning for large volumes fault shape classification

The increasing complexity and density of modern memory technologies, coupled with the continuous evolution of failure modes during production, present significant challenges in the classification of memory fault shapes. Traditional methods, such as heuristic algorithms and earlier Artificial Intelligence (AI)-based approaches, often struggle to address these challenges effectively. Specifically, they face limitations in computational efficiency, scalability to large memory sizes, and adaptability to novel failure patterns. As memory sizes grow and production volumes increase, the need for a robust, scalable, and accurate fault classification methodology becomes critical, particularly in safety-critical applications like automotive systems.

This section introduces a novel machine learning-based framework designed to address these challenges by leveraging Convolutional Neural Networks (CNNs) for fault shape classification in large memory arrays. The proposed approach integrates

pre- and post-processing steps to enhance scalability and efficiency, enabling the classification of known fault shapes while also identifying novel, previously unseen failure patterns. By treating memory failure bitmaps as images, this methodology capitalizes on the strengths of modern image classification techniques, offering significant advantages over traditional solutions.

The section begins by outlining the limitations of existing methods and the motivation for adopting a CNN-based approach. It then provides a detailed description of the proposed framework, including the pre-processing phase for memory tiling, the CNN-based classification process, and the post-processing phase for reconstructing diagnostic information at the memory level. Additionally, the methodology incorporates advanced techniques, such as OpenMax, to detect and handle outliers, ensuring adaptability to evolving failure modes. Experimental results demonstrate the effectiveness of the proposed approach in terms of accuracy, computational efficiency, and the ability to detect novel fault shapes, making it a promising solution for large-scale memory fault classification in modern production environments.

3.5.1 Proposed Approach

The presented approach aims to effectively and efficiently process modern memory failure bitmaps; the analysis result is the classification of the shapes of the failures occurring in a memory bank. Similarly to [9], the proposed methodology uses an Artificial Intelligence framework, specifically a (CNN), joined with a pre- and a post-processing phase. Section 3.5.1 provides an extended summary of the proposed framework. Because the memory size is reaching huge areas and failure modes evolve continuously along production, methods like [10][9] cannot address the problem properly in current days due to computational time and failure shapes characteristics, respectively.

Fig. 3.52 visualizes the proposed flow, which effectively tackles previous approaches' [10][9] difficulties. To the best of our knowledge, this is the first approach facing scalability of classification towards huge memory sizes and automatic identification of unclassified failure shapes.

The pre- and post-processing steps surrounding the classification phase based on CNN permit solving the scalability issue. The memory area is divided into "tiles" in the pre-processing phase. Such tiles are obtained by subdividing the memory area into many squares or rectangles. A tile size selection strategy is detailed in section 3.5.1, which also helps select the most proper CNN engine among the ones available today. Empty tiles affected by no faults are suddenly removed and will not be considered anymore for classification. Once the CNN classifies every meaningful tile, the post-processing step re-assembles the diagnostic information at the failure bitmap level by an algorithmic solution, described in details in section 3.5.1.

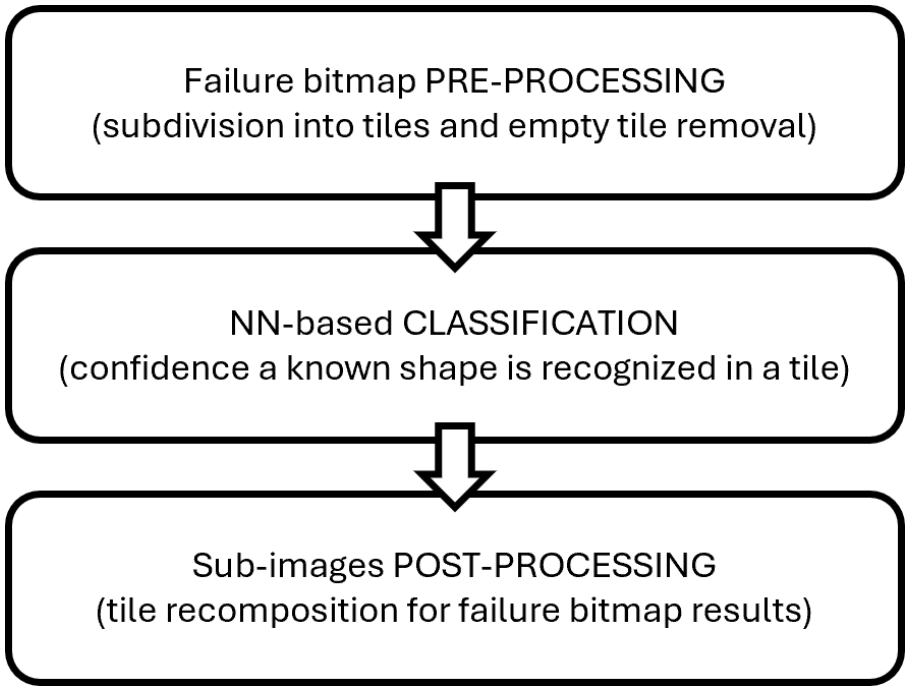


Figure 3.52: Overall view of the proposed flow

Concerning novel failure shapes that could appear during production, the proposed approach overcomes this difficulty by using the openMax methodology [43] as explained in detail in section 3.5.1.

Overall, this research work shows that treating large volumes of failing memory as images provides significant advantages over existing traditional solutions using heuristic nested-loop-based algorithms [10] and also behaving better than previously proposed methods based on Artificial Intelligence methods [9]. The illustrated strategy better explores the trade-offs between:

- *accuracy* and *confidence* in the classification of the failing memory bitmaps into known shapes
- *time* required to classify the failing memory bitmaps;
- ability to detect *new unknown failing shapes*.

Summary of the framework

Fig. 3.53 illustrates the overall flow proposed in this thesis with an example. Starting from the top of the figure, a failure bitmap is visualized. This specific bitmap is artificial and a product of a graphical cosmetic to ease explanations. In this example, the memory size is relatively small (1000x250 cells) and the failure shapes have been magnified to better stem from the drawings; every red dot in the

memory matrix represents a failed bit. The full list of failing bit coordinates is indispensable to put in place the proposed methodology, thus, an effective failure data collection [24] is a strong requirement for the proposed flow.

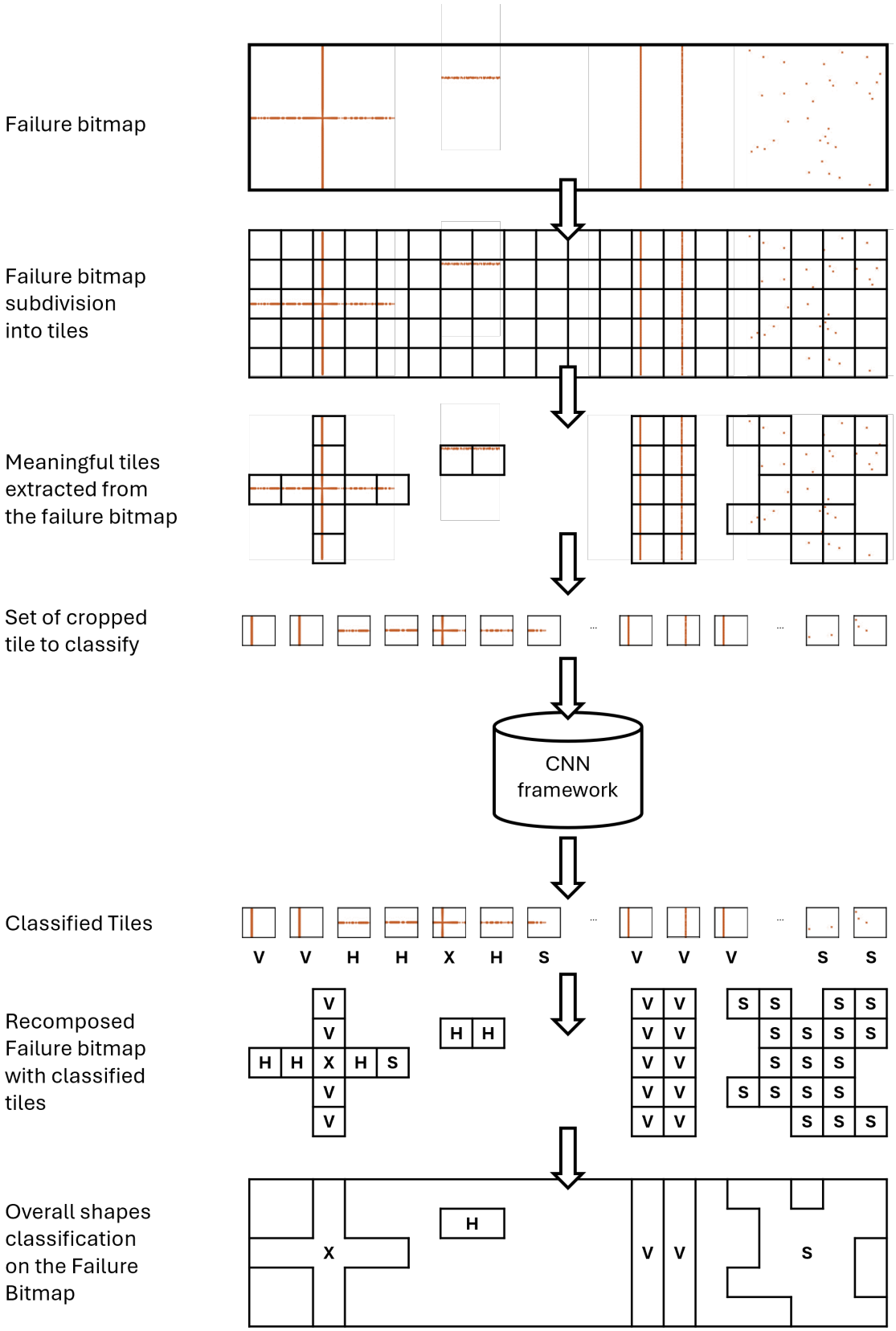


Figure 3.53: Detailed view of the proposed flow

As previously introduced, the failure bitmap is pre-processed before reaching the ML step of the methodology. The failure bitmap is subdivided into the so-called "tiles". The primary aim of this pre-processing phase is to minimize the successive efforts by the CNN.

In the context of huge arrays of memory cells, mass production error rates are usually quite low, and failure shapes are rarely catastrophic (i.e., entire memory blocks failed, large failing clusters); as a consequence, a majority of production failing memories show a few "red dots" surrounded by a lot of "white". Pruning out fault-free zones is also an objective in [9]. The proposed method suggests subdividing the memory into tiles and pruning out from the computation all the tiles with no failure contents. The remaining meaningful tiles are sourced to the CNN framework.

Nevertheless, subdividing the memory matrix into tiles of moderate size also faces some CNN limitations. Current networks implementing image classification [34] cannot process images of too large sizes without incurring extremely high hardware costs and long computational times. Moreover, due to the nature of failure bitmaps where failing shapes to be classified are very small compared to the memory area, the accuracy of the classification may be difficult to ensure. It turns out that the optimal tile size needs to be identified in conjunction with the selection of CNN to utilize. How to select the optimal tile size and the proper CNN to tune the cost function of the proposed method is further discussed in subsection 3.5.1.

Once the set of meaningful tiles is extracted per every memory showing failures along the mass production flow, each one is separately passed to the CNN as a feature for classification. The classification into labels is determined by the training applied to the CNN. The creation of a proper dataset and the fine-tuning of the CNN abilities are described in detail in subsection 3.5.1.

In the example reported in Fig. 3.53, some shape categories are selected among the possible ones. For this explanatory case, only vertical (V), horizontal (H), cross (X), and sparse (S) cases are considered. Additional shapes can be added if they represent significant failure modes and should be monitored. In the explanatory example, all shapes included in the selected tiles have been correctly classified, as it is quite the expected and usual scenario with a high classification confidence level. Nevertheless, the CNN may encounter unexpected shapes (e.g., a combination of shape types in the same tile or odd shapes due to scratching while handling wafers) that need a separate classification. Subsection 3.5.1 provides more details about identifying novel shapes.

After CNN classification, the post-processing phase intends to return to memory from the tile level. Classified tiles are re-assembled according to the original memory layout. This step is based on an algebra of shapes. Adjacent tiles are merged to synthesize diagnostic information at the memory level according to precedence rules (i.e., two horizontally adjacent tiles showing horizontal shapes merge into a single horizontal shape at the memory level), as detailed in section 3.5.1.

Optimal tile size and CNN selection

An important part of the proposed method builds upon the subdivision of the failure bitmap into "tiles". This is a pre-processing step that intends to make the successive computations lighter.

The first product of cropping sub-regions of the memory is the easy pruning of fault-free areas, which normally cover most of a failed memory surface. The Neural Network (NN) processes the remaining "meaningful" tiles, only. This is an effective and low-cost alternative to adopting AI methods that implement pooling [44] at high equipment or CPU/GPU time costs.

The size of the tile plays a significant role in optimizing the proposed strategy's cost. Even though large tiles could seem better because leading to fewer computations, CNN costs per image might be extremely elevated in terms of CPU/GPU time or RAM requirements if the size of a tile is above a certain dimension. Similarly, many small tiles will require low computational effort for the single tile, but a higher number of CNN elaborations. Looking for the best trade-off among the size and number of tiles then is crucial to maximizing the benefit of using CNN for quick classification of failure shapes.

The optimal tile size selection primarily depends on the expected failure shapes and their dimensions. For example, suppose the assumption is having sparse failures homogeneously distributed over the surface of the memory. In that case, it is possible to trace a curve showing the number of tiles needed to cover all defective memory bits at the varying tile sizes. Fig. 3.54 shows the trends for a set of memories showing sparse failure constellations. In this explanatory case, this figure spans from seven to almost 100 thousand failures in a memory core of 8960x1056 cells following an exponential formula. For each number of failing bits detailed in the legend in Fig. 3.54, 40 memory cores were simulated and injected with heavenly distributed fails. In the figure, groups with the same number of injected failing bits are represented with the same color going from purple (less faults) to red (maximum number of faults). The ordinate axis reports the squared tile side size in bits (or pixels), while the abscissa axis reports the relative number of meaningful tiles. The average trend is highlighted with black squares.

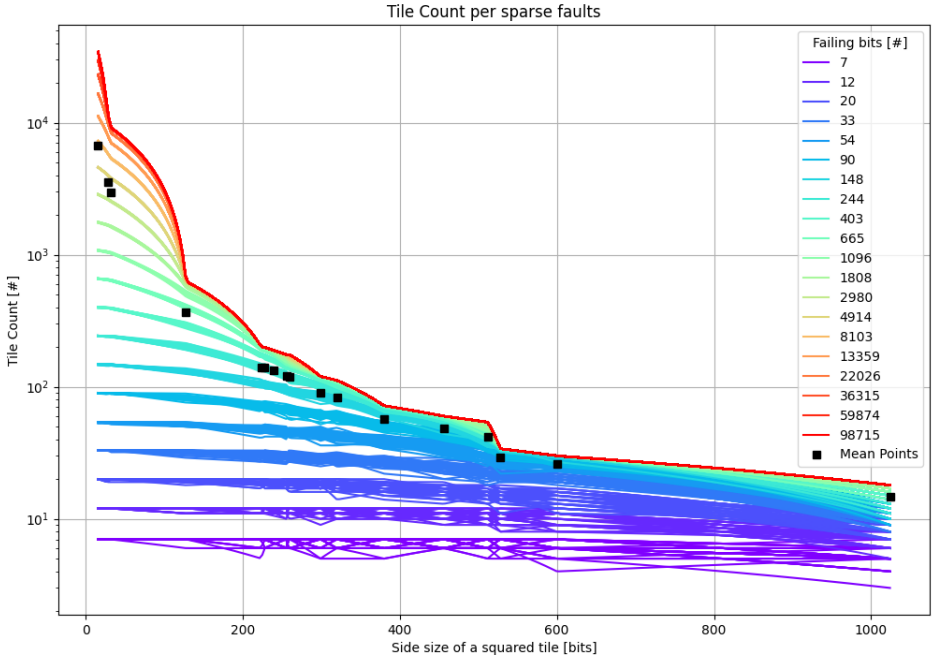


Figure 3.54: Detailed view of the relationship between tile size and number of significant tiles in case of sparse failure cases

The number of meaningful tiles may vary greatly depending on the tile size. The number of meaningful tiles is lower when the expected failure shape is more localized and solid than sparse shapes (i.e., a horizontal or vertical failing line, or a cluster of failing bits). Using real data from production is the most appropriate decision.

The CNN accuracy and confidence level are also important ingredients for the selection of the optimal size of the tile, as many CNN engines are available today for image classification. Which one of the available CNNs to select is a question that can be answered by exploring the trade-off between (a) the size of the tile, which implies the computational efforts for a selected CNN, (b) the number of meaningful tiles, and (c) the accuracy and confidence of the selected neural network.

Therefore, we suggest a two-step strategy to determine the optimal tile size in conjunction with the CNN selection:

1. exploring tile size vs number of tiles trade-off: this analytical phase can be based on real data coming from the production flow, or data being artificially generated (but showing the same failure characteristics as the production does)

2. selecting the CNN that better fits the expectation of having very high classification accuracy in the shortest time possible.

Starting from point 1, it is logical that the tile size impacts the computational time required by the CNN to compute the result. Maybe, higher computational costs are paid for a few large tiles, than having more small tiles; given that every specific CNN exposes its own computational cost, selecting a tile size from graphs like in Fig. 3.54 must be connected to CNN costs. Thus, point 2 becomes part of the tile size selection criteria; the better trade should be found among the number of tiles, the computational cost for the CNN to crunch a single tile, and the accuracy of the CNN.

All these considerations bring us to point 2; the selection of the CNN to deploy becomes contextual with the selection of the tile size; accuracy and confidence perspectives of the possible alternatives of CNNs to be used become factors to consider contextually to the tile size.

The following is a possible way to synthesize a rank for a combination of tile size/selected CNN. Given a tile size corresponding to the image size managed by a CNN_i from a pool of N CNN candidates, it is possible to compute the cost as a simple multiplication of CNN parameters (e.g., impacting on the CPU time) by the number of tiles as in formula (1).

$$Cost^{CNN_i} = Parameters(CNN_i) * \#Tiles \tag{3.14}$$

Then, a weight is added to the computation, as in formula (2), corresponding to the expected accuracy of the selected network, which could be derived according to the vast literature on this topic. Accuracy expectation is a value that varies between 0 and 1, where 1 is the maximum value of accuracy corresponding to 100% of correct classifications.

$$WeightedCost^{CNN_i} = Cost^{CNN_i}/Accuracy(CNN_i) \tag{3.15}$$

Ranking from low to high Weighted Cost the pool of CNN frameworks permits operating an effective selection. In the case of similar results for different networks at the top of the rank, the confidence expectation of the CNNs that stem from the computation can be a discriminating factor.

Dataset creation

Once a specific NN is selected, and the tile size chosen contextually, an instructor trial is necessary to tailor the CNN weights. A carefully crafted Dataset must be developed by leveraging as much as possible on real failing bitmaps and augmented data if necessary to avoid class imbalance. In simple words, a Dataset is a "large enough" bunch of cropped tiles, whose content should be very representative of the most common and expected failure shapes. Nevertheless, it considers as valid

shapes, such as vertical, horizontal, etc, also not continuous failure shapes, which is tricky to implement in an algorithmic way [10]. Examples of non continuous shapes are in the failure bitmap of figure 3.53.

As already shown in [9], it is first of all necessary to list the type of failure shapes that are expected to be dominant in the production. Table 3.15 reports a list of basic failure shapes as an explanatory example. The list of shapes may be extended or a shape category split into many ones depending on the considered technology.

Table 3.15: Description of known failing shapes in memory bitmaps.

Label	Description
HORIZONTAL	Horizontal shape*
VERTICAL	Vertical shape*
CROSS	Combination of an horizontal and a vertical shape
SPARSE	failing bits are spread over the memory surface

Once the failure shapes to target are identified, the database of shapes needs to be populated with a sufficient number of tiles containing cases of the failing shape. How to select the right tiles to add to the dataset is a crucial task to reach the accuracy level expectations.

Ideally, this selection process should be fully based on the extraction of real shapes from real faulty devices. This is not always feasible for some practical reasons, such as it is not easy and very time-consuming if a human person is required to search "by hand" for the right amount of tiles.

To partially solve this issue without fully resorting to algorithmic solutions like in [10], it is suggested to exploit some additional information that may be returned from a failing chip under test. Failure data collection approaches like [16][25][24] compress or compact the failure bitmap of a failing device in order to minimize on-chip memory requirements and the transfer time from chip to tester. Compact or compressed bitmaps, which are fast to transmit, are then "unpacked" offline and ready to be saved in the cloud and, individually or statistically, analyzed. Compaction or compression information, which continues to be available after bitmap reconstruction, can be used to roughly classify the failure shape and guide the test engineer to a faster and more fruitful selection than before. Figure 3.55 illustrates a possible ecosystem for a test engineer devoted to select meaningful tiles to train the CNN.

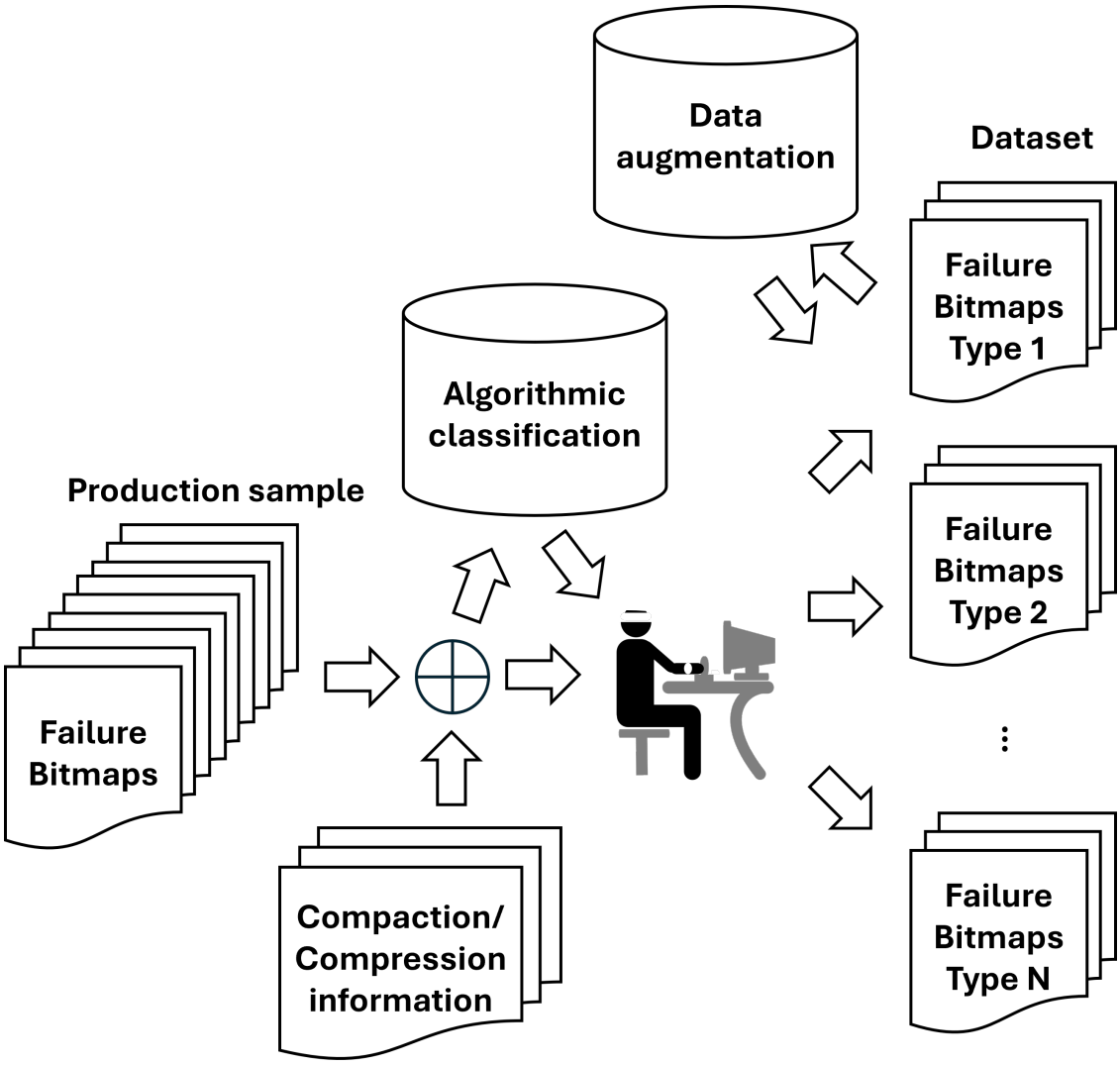


Figure 3.55: Illustration of the Dataset creation flow

Nevertheless, in real cases, the number of failure cases for one shape category could be abundant and necessitate a selection, while, in the most unlucky case for a strong dataset creation, for another the number of real cases is low. By training a ML classifier with more training data, it is possible to enhance the performance of the proposed model [45] and obtain better generalization on in-domain samples.

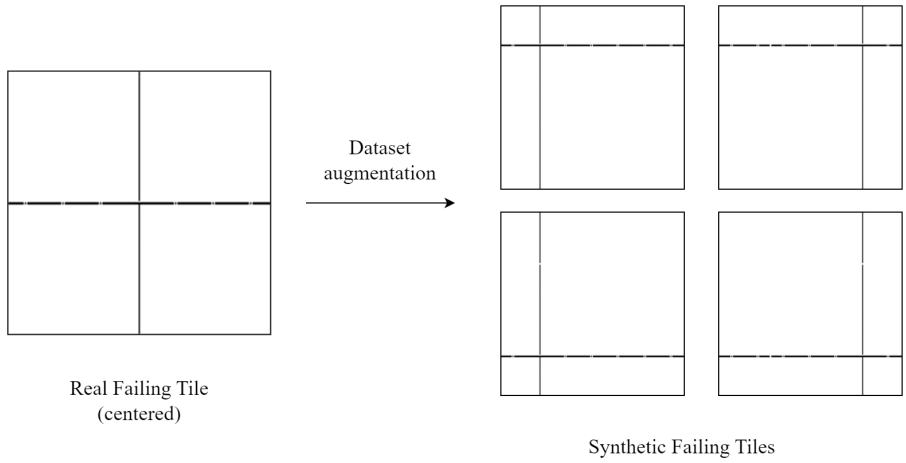


Figure 3.56: Example of dataset augmentation by applying shiftings and translations.

Given such a realistic unbalanced distribution of real failing tiles, it is often necessary to resort to extra artificial data. Data augmentation is a popular technique [46] consisting of artificially generating new data sets from existing ones with synthetically created samples.

Among the existing augmentation techniques [45], relevant transformations closest to the real case were applied. Starting from the images of the available dataset, new synthetic images have been obtained by using shiftings, translating, cropping and clipping [47].

An example of image augmentation is illustrated in Figure 3.56. For the sake of comprehensiveness, rotation has not been used as it is unrealistic to get rotated bitlines (or other shapes) in failing memories.

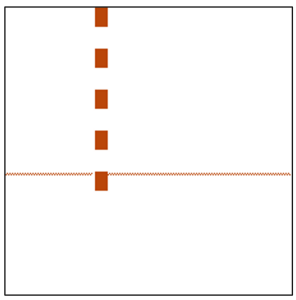
NN-supported Diagnosis of novel failure shapes

One of the key advantages of the proposed approach is the detection of known failing shapes (according to Table 3.15) along with the filtering of *novel* unknown shapes, to be carefully analyzed by experienced engineers. This class of failing shapes is also referred to as outliers, to be aligned with the terminology of the deep learning world. The problem of identifying novel failing shapes falls into the problem of identifying Out-Of-Distribution (OOD) samples from in-distribution (ID) ones [48]. In other words, ID samples correspond to the training data that have been used to instruct the NN to recognize a given set of failing shapes. OOD samples are failing shapes never seen by the NN.

Among the different OOD detection techniques, confidence-based approaches [49] are widely used, effective, and simple at the same time. In the literature, it is known that the prediction probability of incorrect and out-of-distribution samples

tends to be lower than the prediction probability for correct (ID) ones [50]. However, it is not convenient to rely solely on the confidence of neural networks, as they tend to be overconfident even in the case of incorrect predictions: this phenomenon is also known as *agnostophobia* [51] and is even more crucial for very deep neural network architectures.

Therefore, to identify outliers (i.e., novel memory failure shapes in our case of study), a methodology to estimate the probability for a given input belonging to an unknown class was integrated, i.e., OpenMax [43]. To identify OODs, OpenMax compares the logit of an inference (i.e., the penultimate layer of the network before the Softmax) with the average logit recorded during training for ID classes. More in details, OpenMax calculates the Mean Activation Vector (MAV) for each ID class. The MAV for a specific class is determined as the average logit of all *correctly* classified samples within the training set. The greater the distance between a sample's logit and the MAV of its predicted class, the higher the likelihood that the input is OOD. Hence, OpenMax substitutes the Softmax activation function with the OpenMax function. At inference time, the distance between the computed logit and the MAV distribution is used to revise the logit and add an *open-set class* (OC) score, yielding the *OpenMax vector*, namely a new score vector including a new class (i.e., the *unknown-unknown class*) to collect all OODs $f(x) \in \mathbb{R}^{|C|+1}$, where C corresponds to the ID classes. Therefore, OpenMax provides probabilities that support explicit rejection when the unknown class has the largest probability.



Shape probability	Confidence
HORIZONTAL	50%
VERTICAL	35%
CROSS	12%
SPARSE	3%

Figure 3.57: Example of an outlier failing shape

Figure 3.57 reports an outlier example coming from production.

Post-processing algorithm to summary from tile to memory level

Once each tile is classified, it is crucial to reconstruct the failure information at the memory level from the classified tiles. For statistical reasons, as desirable to monitor a mature mass production process, it is valuable to have volume data organized by failure shape at the memory level, while more emphasis is put on tile-level "micro" failures, including novel failure identification that just necessitates tile information.

This section aims to illustrate an effective algorithm that crunches tile-level information and returns a list of shapes classified at the memory level.

The algorithm inputs the list of classified tiles T in memory and returns the list of shapes S identified at the memory level. Every element in list T transports a few fields:

- X, Y: the coordinates of the tile inside the memory, values available from the pre-processing stages;
- $tClass$: the classification of failure into a category of shape at tile level, a value inherited from the NN elaboration;
- $mClass$: the classification of failure into a category of shape at memory level, corresponds to the tile level classification before post-processing;
- $mark$: it is an algorithmic boolean field necessary to optimize the computation, reset to False at time zero.

Algorithm 4: Tile visit to synthesize memory level shape statistics

```
S ← [];
procedure getMemoryShapesFromTiles for each tile t in T do
    if not t[mark] then
        S.append(adjacent(t));
print(S);
end procedure
procedure adjacent(c) if not c[mark] then
    c[mark] ← True;
    for each tile t in T do
        if not t[mark] and t.isAdjacent(c) then
            s ← adjacent(t);
            m ← merge(c[mClass], s);
            if m = ILLEGAL then
                S.append(s);
            else
                c[mClass] ← m;
return c[mClass];
end procedure
getMemoryShapesFromTiles();
```

The listing 4 describes in a Python language style how to explore the tile list in a kind of post-order breadth-first visit and uses the mark field to optimize the exploration. Along the visit, the search explores the memory shapes recursively and the shape computation is elaborated along the backward phase of the algorithm.

Every tile in the list T is requested to look recursively to the neighborhood; adjacent tiles are identified by a function called $isAdjacent(t1, t2)$ that compares tiles' coordinates eventually accepting a ratio as an additional parameter (i.e., to define how far could be the neighbor). Every tile that is reached by the search is marked as visited and not considered anymore successively. The recursive exploration of the neighborhood ends when a tile has no unexplored failing neighbors left; for the leaves of the traversed tree, the returned value is the shape classification at the tile level. By going backward recursively level by level, the recursive function caller receives a shape from every opened branch and uses the merge function to aggregate the current tile shape with the shape coming from the successive tiles.

The merge function is based on an algebra of shapes that determines the resulting shape when combining the shapes from two tiles. A partial summary of a possible algebra is reported in table 3.16. The current classification of tile c and its neighbor tile t are compared and the relative position of the tiles is also included in the operation (i.e., A/B stands for Above or Below, while L/R for Left or Right neighbor). The result is often an updated shape, but it can also fall into ILLEGAL configurations. At memory level, the classification may be expanded to more categories, such as it is shown in table 3.16. In this explanatory case, new shapes are introduced at memory level, like double and multiple vertical shapes.

Table 3.16: A possible shape merge algebra, limited to compute about vertical, cross, and sparse tile cases, returning memory level classification.

c	**t**	**relative position***	**m**	**notes**
V	V	A/B	V	single vertical
V	V	L/R	VV	double vertical
V	VV	L/R	MV	multiple vertical
V	H	A/B	ILLEGAL	incomplete cross
V	H	L/R	ILLEGAL	incomplete cross
V	X	A/B	X	large cross
V	X	L/R	ILLEGAL	potential outlier
V	S	A/B	V	incomplete tail
V	S	L/R	ILLEGAL	potential outlier
X	S	-	ILLEGAL	potential outlier

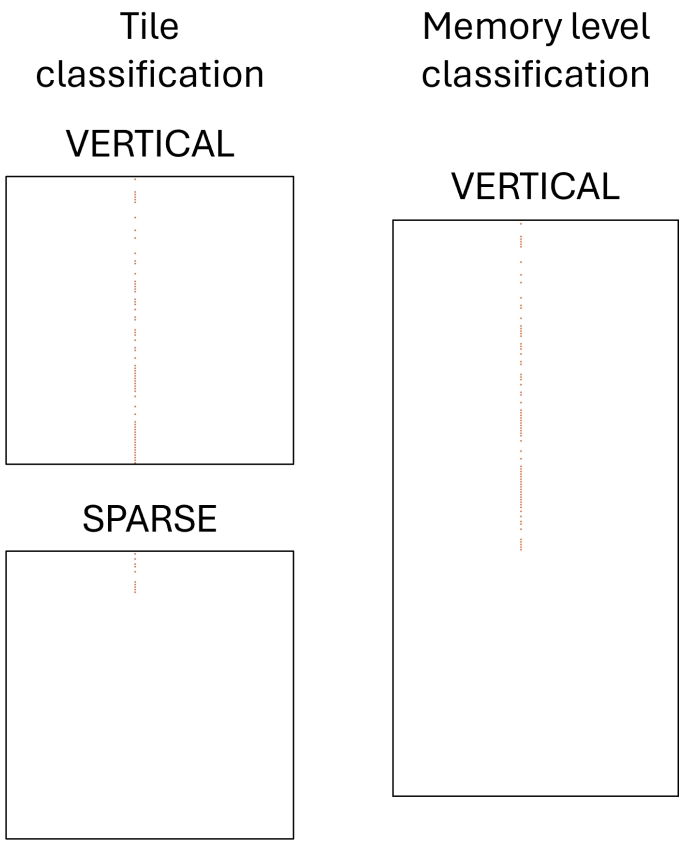


Figure 3.58: Example of merge that corrects a wrong classification at tile level

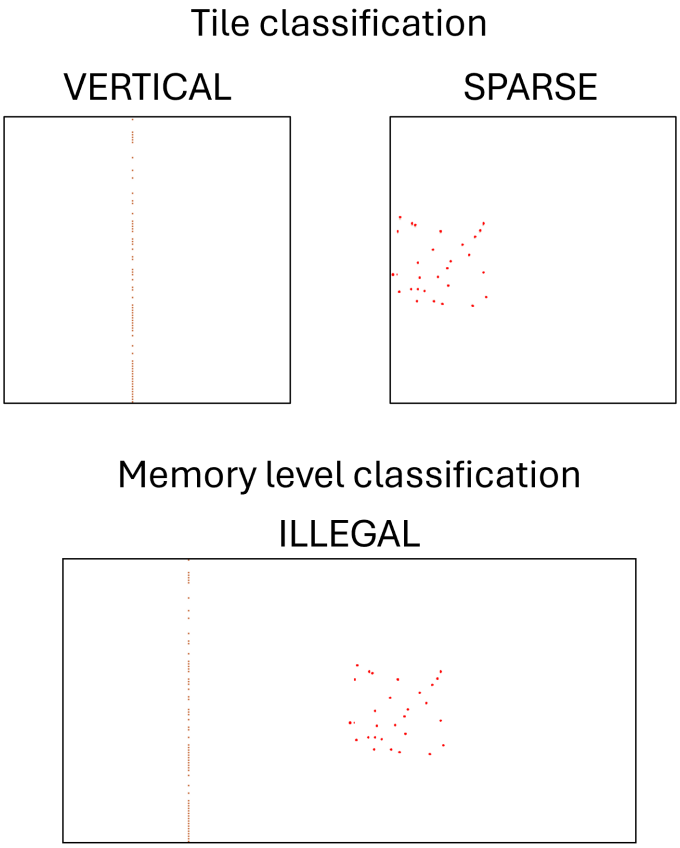


Figure 3.59: Example of an ILLEGAL merge that brings attention to a potential outlier at memory level

The merge operation is also beneficial in correcting some CNN wrong classifications that sometimes happen because of the arbitrary cropping of tiles. As in the example provided in figure 3.58, a tile could be misclassified but the merging algorithm can correct many of these cases. An ILLEGAL merge case may also emerge, which could pinpoint an outlier at the memory level. An example of an ILLEGAL merge of a VERTICAL and a SPARSE tile horizontally adjacent is depicted in figure 3.59.

3.5.2 Experimental Results

This section will introduce the experimental case of study (Section 3.5.2) and discuss the implementation choices (Sections 3.5.2-3.5.2) and will present the obtained results, along with a careful analysis of the performance and the costs of the proposed approach (Section 3.5.2).

Case of Study

For the case study, we selected an Automotive SoC manufactured in very large volumes by Infineon. The chip includes more than ten Megabytes of Non-Volatile Memories (NVM) which are distributed among several memory banks located on the SoC surface.

Such Automotive SoCs are safety-critical and must have a high level of reliability. Thus, a quick and accurate classification of failing shapes is very important to ensure a sufficient quality level. The NVM test flow is composed of many test steps that bring the Devices Under Test in several combinations of temperature, system frequency, voltage range, memory access time, etc. [1].

Along the memory test runs, which are applied by a hardware-firmware combination (e.g., an MBIST behavior is controlled by a CPU), failure information is stored on-chip in the available volatile memory, which is not involved in the test, and transferred to the ATE at the end of the test execution or before releasing the chips from the current ATE insertion. To face the limitations in terms of on-chip memory requirements that may arise when the number of failing bits is large, failures' coordinates are stored in a compact format by exploiting an extra software ingredient described in [24]. The algorithm in [24] processes the faulty bits as soon as they are discovered by the MBIST, looking for possible aggregation of many faulty bits together to save space without losing information. As detailed in [24], the algorithm encodes the faulty coordinates in a lossless compact format, for example compacting a set of adjacent faulty bits into a data structure including the first and last coordinate of the faulty "slice", the direction (horizontal or vertical) and a color to label different faulty pattern (i.e., continuous or regularly intermittent failure between the first and the last fail in the slice). Such extra information coming from a compaction process, mainly necessary to save on-chip memory, is then greatly beneficial for the dataset creation.

Optimal Tile Size and CNN selection

Each NVM memory bank is further divided into physical sectors. Every physical sector is tested individually; physical sectors belonging to different banks are tested in parallel. This strategy enables the full test of the NVM banks in a shorter time and, joined with the test parallelization on multi-site ATEs, permits the production of huge volumes of devices.

Physical sectors of the considered chip correspond to an area of 8,960x544 bits corresponding to around 0.6MB of capacity. As motivated in section 3.5.1, it is hard for any CNN to process such a large failure bitmap as a single image. As illustrated in section 3.5.1, a viable solution is based on dividing it into tiles, which could be rectangular or squared, as in this case.

Fig. 3.60 shows the trends of the number of meaningful tiles vs. the tile size for production shapes extracted from corner lots, encompassing a significant number

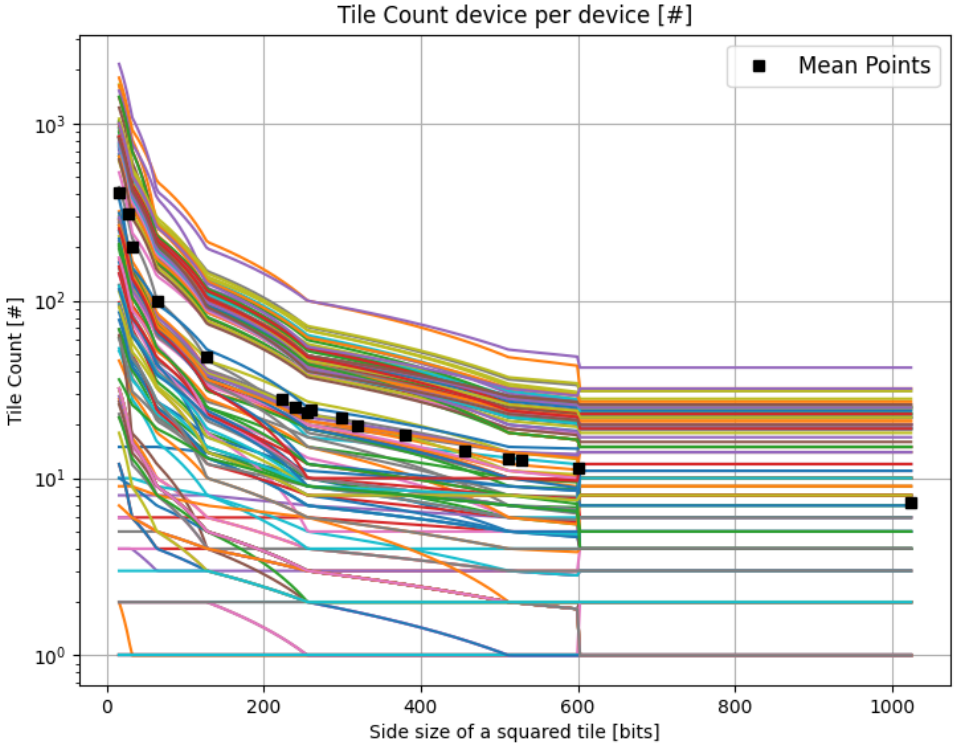


Figure 3.60: Average trend of number of tiles over tile size in case of horizontal, vertical, cross, and sparse failure constellations from production

of dies from a large production volume. Selected failing memories are expected including many instances of failures deriving from the same root cause and valuable to train a CNN properly.

Based on these production data and a list of promising CNN architectures available in the NN arena, formulas (1) and (2) introduced in section 3.5.1 are used to extract a rank of the possible tile size/CNN combinations. Table 3.17 shows the top elements of a longer list based on a deep search for CNN architectures and their abilities in terms of image size, CNN parameters, and accuracy, combined with the production data analysis. The Cost is computed as the number of expected meaningful tiles multiplied by CNN parameters, which are used here as a surrogate of the CPU time needed by a CNN to complete the analysis of a tile.

ResNet18 looks to be the most promising CNN engine to use for the illustrated case of study. It is then considered as the selected CNN. ResNet-18 is a widely popular network architecture: today, in the task of image processing (classification, segmentation, object detection), ResNet-like architectures are considered as

Rank	CNN	Tile Size [bits]	AVG Tiles	Parameters	Cost	Expected Accuracy	Weighted Cost
1	ResNet-32	32x32	202.01	0.46	93.73	90.90%	1.03
2	ResNet-18	224x224	28.01	4.00	112.05	99.50%	1.13
3	EfficientNet-B0	224x224	28.01	5.30	148.47	77.10%	1.93
4	InceptionV1	299x299	21.88	6.80	148.78	69.77%	2.13
5	DenseNet 40	32x32	202.01	1.10	222.21	94.73%	2.35

Table 3.17: Ranked list of the top 5 CNNs

the reference for optimizing the hardware computational requirements [52], or investigating the effects of various activation functions [53]. Indeed, they are quite popular, having been the first architectures in the ImageNet Challenge to achieve a top-5 error rate lower than that of human performance [54].

Dataset Creation

Having defined the *processing* size of the failing bitmaps (i.e., failing tiles of 224x224 bits), the next step corresponded to the creation of the dataset for training an ML model.

The list of considered shapes in the experimental setup is the following:

- BITLINE/WORDLINE: it is a vertical/horizontal shape that could be also subdivided into full, partial, or intermittent
- TWO BITLINES: Two vertical shapes are located far from each other, at a critical distance (e.g., such as scrambling-significant distance)
- MULTIPLE BITLINES: multiple vertical shapes are located far from each other (not necessarily at a regular distance); in the following, the distance is fixed to 32 bits
- MULTIPLE ADJACENT BITLINES/WORDLINES: multiple vertical/horizontal shapes that are located close to each other
- WORDLINE X BITLINES: it is a cross created by a Wordline and a Bitline incident to each other
- SPARSE: failing bits are spread over a region of the memory.

A further shape category, which does not take part in the training process of the CNN, is the Outlier category, which includes all "odd" shapes coming from production (e.g., those not falling into the defined labels). All Wordline- and Bitline-oriented categories consider non-continuous shapes.

Leveraging on an experimental setup including (a) the complete and lossless failure bitmap of every failing physical sector, (b) compaction information deriving from the approach in [24],(c) a failure bitmap visualization tool [55] and (d) some basic algorithmic functions recognize failure shapes (at high cost), a test engineer

selected real failure shapes from the same production lots used to determine the optimal combination of tile size and CNN to adopt.

From such failure data, a total of 35,180 real failing tiles of size 224x224 bits have been extracted and analyzed to be categorized into different failing shapes. This process was performed by exploring both manual effort and loop-based heuristic algorithms that count the number of failing rows, columns, and bits, with the intent of classifying them into known images. Table 3.15 illustrates the percentual amount of sample tiles found along the partly automated selection process.

Table 3.18: Final composition of the dataset, including real images and extra data obtained by resorting to data augmentation. Each image is made by 224x224x1 pixels.

Labels	**Total Labeled Images**	**Selected Tiles from real chips [%]**	**Augmented data [%]**
BITLINE	5,000	100	0
WORDLINE	5,000	100	0
TWO BITLINE	5,000	40.58	59.42
MULTI BITLINE 32	5,000	11.24	88.76
MULTI ADJ BITLINE	5,000	16.26	83.74
MULTI ADJ WORDLINE	5,000	100	0
WORDLINE X BITLINE	5,000	22.58	77.42
SPARSE	5,000	100	0
TOTAL	40,000	-	-

The analysis of those 35,180 failing tiles revealed an unbalanced trend across the different classes. Figure 3.61 reports statistics on the occurrence of the different failing shapes in real failing memory physical sectors. As emerging, more than the 25% of failing tiles are classified as sparse. The other majority of tiles belongs to the bitline, wordline, and multi adjacent wordline (∼16%, ∼20%, ∼22%, respectively). The union of those 4 categories surpasses ∼85%. The remaining 4 classes occur with a very low frequency.

The trend shown in Figure 3.61 points out that the likelihood of occurrence of some failing shapes (i.e., bitline, wordline, sparse, multi adjacent wordline) is much higher than others. To address this imbalance, synthetic tiles were generated for underrepresented failing shapes, as above-described. Therefore, given the number of the available real failing tiles, a bound of 5,000 images has been selected for each class of the dataset. Table 3.18 also reports the final composition of the dataset, including percentages of real images and synthetic ones, if any. This process guarantees the same number of samples for each category in our dataset. Overall, a total of 40k images were collected for classifying failing shapes in large volume

failing memories.

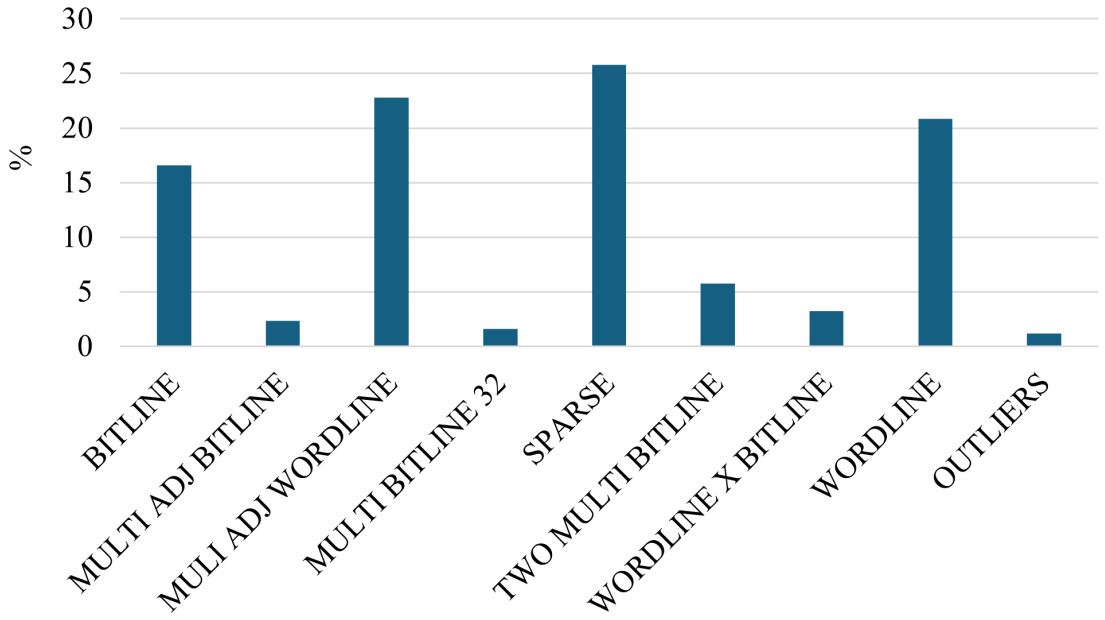


Figure 3.61: Percentage of failing shapes (i.e., classes) obtained from the analysis of 35,180 failing real tiles.

The reader can notice from Figure 3.61 that over 35,180 failing tiles analysed, the 1.17% has been classified as "outliers", meaning that the obtained shape does not fall into any of the known faulty shapes. This category is not included in the final dataset (as reported in Table 3.18). The outlier detection is a crucial point in the proposed methodology. Instead of incorporating different unknown shapes in a single class of "outliers" (that would have made the training more complex and with the risk of non-convergence), the proposed approach relies on the OpenMax methodology for outliers' detection, as described in Section 3.5.1.

Accuracy, performance, and costs

As previously shown, ResNet-18 was selected, trained and deployed on our database of failing shapes of NVM.

We have trained ResNet-18 on PyTorch (a popular tool for neural network inferences) for 200 training epochs, with a batch size of 64, a learning rate of 0.1, and a weight decay of 0.001. The final model achieves 97.81% of accuracy in classifying the 8 classes (Table 3.18) of the test set, and 97.64% on the validation set, for about 4 million parameters. In line with the tile size discussion, the first layer of our implemented CNN receives B&W images of size 224x224x1.

Table 3.19 summarizes the tile- and memory-level accuracy over a significant base of about 12K failing chips from production lots. Also, it is reported the confidence level of the CNN for tile classification. Table 3.19 reports also the accuracy for outlier shapes; this result is obtained by integrating OpenMax to the ResNet-18 model as detailed in Section 3.5.1.

Label	**Tile-level Accuracy [%]**	**Memory-level Accuracy [%]**
BITLINE	99.33	99.39
WORDLINE	99.70	99.80
TWO BITLINE	100.00	100.00
MULTI BITLINE 32	99.70	100.00
MULTI ADJ BITLINE	96.67	98.67
MULTI ADJ WORDLINE	97.94	97.94
WORDLINE X BITLINE	90.91	97.91
SPARSE	99.80	99.90
OUTLIERS	91.89	91.89
Total	99.18	99.50

Table 3.19: Proposed method results about accuracy at tile (before post-processing) and memory (after post-processing) level, and the average confidence of the classification by CNN at tile level

The proposed method looks very accurate and fast in the classification of known shapes and is also effective in identifying new, unknown failure shapes.

Method	**Accuracy [%]**	**CPU Time [h]**
Algorithmic [10]	99.92	3.66
ANN-Based [9]	90.80	1.76
Proposed	99.50	0.51

Table 3.20: Accuracy and cost comparison with ANN [9] and algorithmic [10] methods applied to classify the base of over 12K failing memories

A comparison with two approaches is reported in table 3.20; the first one is an algorithmic solution, which is proposed in [10], the second is the ANN-based approach illustrated in [9]. It is reasonable that the algorithmic approach reaches the highest accuracy, but requires significant time to perform the classification. The ANN-based is the fastest, with the lowest accuracy among the compared methods.

Finally, the proposed approach shows an exciting trade-off between accuracy and time. It is quite accurate, losing only 0.42% from the algorithmic, and it is also slightly faster than the ANN-based approach in which pre- and post-processing is particularly heavy [9]. Results have been patiently validated manually by the test engineer appointed to cross and verify results from the different approaches. The availability of the same test engineer is suggested to monitor production failure classification results, supported by a bitmap visualization tool and guided by the automatic classification, giving primary attention to outliers.

3.6 Novel Approach for Extracting Memory Design Configurations

Modern memory systems are increasingly exposed to radiation-induced faults. These faults, caused by charged particles interacting with memory cells, often manifest as Multiple Event Upsets (MEUs), where multiple bits fail simultaneously. Analyzing these MEUs provides a unique opportunity to infer the underlying memory design configurations (MDCs), which are crucial for understanding memory behavior and improving fault tolerance.

Reconstructing MDCs from MEUs, however, is a complex task due to the wide variety of possible memory architectures and the intricate patterns of faults generated by radiation. This section presents a systematic approach to address this challenge, introducing an algorithm capable of efficiently identifying the most probable MDCs by leveraging the spatial characteristics of MEUs. By focusing on the proximity of faults and their patterns, the proposed method offers a robust solution for extracting memory parameters and validating memory designs under fault conditions. This work has been published at the IFIP/IEEE VLSI SoC 2023 conference [56].

3.6.1 Proposed Approach

The proposed approach uses an efficient algorithm to extract the MDCs by observing the memory MEUs from irradiation tests. MEUs are generated by the interaction of a charged particle with multiple memory cells. The affected cells can, for example, share a corner or be one in front of the other. For this reason, the algorithm assumes that each set of MEUs is composed of faults near each other, following the principle that a charged particle generates failing patterns (MEUs) such as the ones illustrated in Fig. 3.62. As can be seen, each bit in the failing pattern is next to at least another failing bit in either the same row or column. The proximity of the MEUs is the central assumption of our algorithm that can test multiple possible memory organizations to extrapolate the correct ones and exclude the wrong ones as soon as possible.

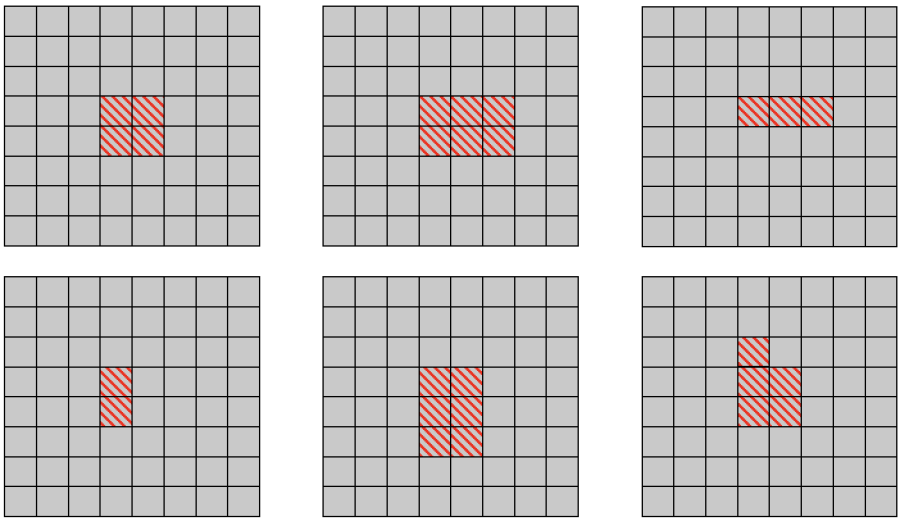


Figure 3.62: A general view of MEUs in memory cells.

Memory reconstruction

The proposed algorithm's aim is based on the assumption that each MEU from the irradiation tests generates faults located near each other, as shown in Fig.3.62 an. These MEUs stem from a charged particle hitting the intersection between 2 or 4 memory cells, causing them to be interpreted as failing bits. So MEU is composed of a variable number of faults. Due to this assumption, the proposed algorithm performs a series of steps for each set of MEUs received:

1. **Guess an MDC**: The aim of the algorithm is set to test all the possible MDCs systematically. It achieves this goal by changing the various parameters one at a time until an exhaustive search is performed. This approach represents a guess by the algorithm that will then test this MDC to discover if it can be the source of the MEUs.

2. **Fault reconstruction**: This reconstruction is performed considering logical addresses of faults and the selected MDCs. As each MEU is made of a certain number of faults, at the output of this step, the algorithm obtains a set of physical coordinates (one for each fault in the MEU set).

3. **Check the contiguity between the reconstructed faults**: Given the fault coordinates, the distance between each fault is checked. The main objective is to find all faults' locations in close proximity. In other words, a group of fault locations is called MEU when:

    - There are at least two fault locations with a Euclidean distance of 1 from each other.

- The third fault location can be added to this faults' locations group if it has a Euclidean distance equal to 1 from at least one of two faults' locations.
- Other faults can be added to the mentioned group if they have a Euclidean distance equal to 1 from at least one of the other faults' locations group. These groups of x and y coordinates should create some patterns, such as the one shown in Fig. 3.62.

For instance, in Fig.3.63, the location of the MEU is compliant with the chosen MDC because the faults are in the same neighborhood. However, as can be seen in Fig.3.64, the chosen MDC is non-compliant because the faults are in different neighborhoods and separated.

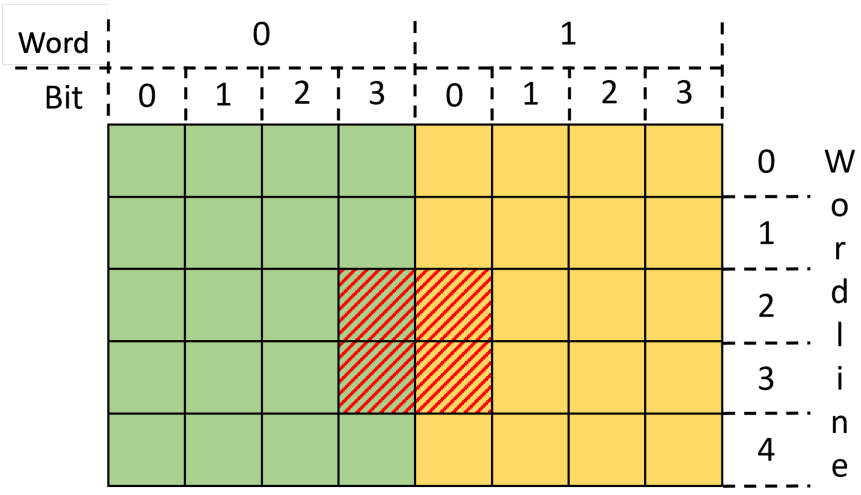


Figure 3.63: A scheme of reconstructed memory with the correct MDC.

4. **Update correct combinations or remove MDC from possible solutions**: If the combination is marked as correct, the algorithm increases a counter that counts how many times the current MDC was identified as correct. Otherwise, no further steps are performed, and the MDC is removed from possible solutions.

The steps from 2 to 4 are repeated for all the MEU sets related to a given memory. The algorithm continues to test until all the MDCs have been tested. Once all the MDCs have been fully explored, the possible combinations are ordered from the most probable to the least probable. The correct memory organization is in the most probable ones. Often the algorithm reports equivalent. The complete algorithm is provided in algorithm 5. This algorithm has a programmer's point of view on the proposed approach.

Algorithm 5: Memory parameters extraction algorithm

```
input  : A list of MEUs. Each MEU is composed of N faults, each with:
         The logical word address of the fault: Addr
         The failing bit inside the word: Q
output: A set of MDCs that can generate the MEUs in the list.
foreach MEU in MEUs list do
    foreach Memory Dimension in Possible Memory Dimensions list do
        foreach Row Mirroring in list do
            foreach Column Mirroring in list do
                Take the Addr and Q and calculate faults' coordinates (n is
                 the number of fault locations): (x_i,y_i), (x_{i+1},y_{i+1}), ...,
                 (x_{i+n},y_{i+n}) And make a group of coordinates by randomly
                 adding one coordinate to the group.
                while At least a fault coordinate is added to the group in this
                 iteration. do
                    foreach Faults' coordinates do
                        if The coordinates have a Euclidean distance of 1 at
                         least with one of the coordinates in the group. then
                            Add the coordinate to the group.
                        end
                    end
                end
                if The group contains all of the faults' coordinates then
                    Mark the MDC as one of the possible solutions.
                end
                else
                    Remove the MDC from the checklist.
                end
            end
        end
    end
end
```

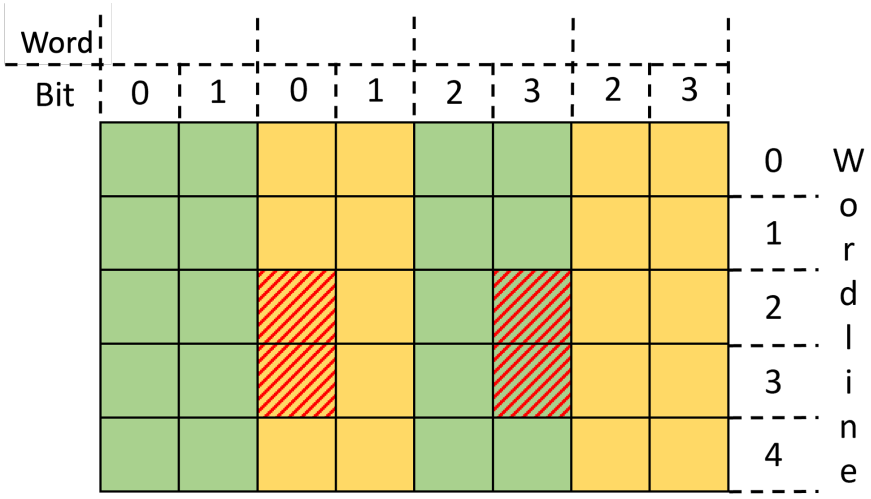


Figure 3.64: A scheme of reconstructed memory with the wrong MDC.

Memory Design Configurations			
	Min	**Max**	**Steps**
Total Number of Words	2	2^{15}	Exponential (Powers of 2)
Words Per Wordline	2	$2^{\log_2(\text{Total number of words})-3}$	
Bits Per Word	2	2^8	
Wordlines	$\frac{\text{Total Number Of Words}}{\text{Words per Wordline}}$		
Wordlines Mirrored Every	2^0(no mirroring)	$2^{\log_2(\text{Wordlines})}$	Exponential (Powers of 2)
Bitlines	Bits per word * Words per Wordline		
Bitlines Mirrored every	2^0(no mirroring)	$2^{\log_2(\text{Bitlines})}$	Exponential (Powers of 2)

Table 3.21: Memory parameters taken into consideration for our experiments

Experimental Results

The research aims at extracting the memory design parameters, and the proposed approach uses the Algorithm 5. The final results can then be used by test engineers to accurately assess the behavior of their programs in case of radiation-induced MEUs. In order to corroborate the functionality of the proposed approach, a complete simulation and validation environment was developed. This environment comprises multiple scripts that perform a series of steps provided in Fig. 3.65 to inject the MEUs, and then reconstruct the memory parameters. The Python script responsible for the simulation and injection of the MEUs is called "supervisor script." This supervisor script:

1. Randomly select an MDC made of all the parameters described in the previous

section

2. Randomly select multiple MEU shapes and their physical location inside the chosen memory. The shapes are selected with respect to the ones described in Fig. 3.62
3. Calls a C++ memory creator script that simulates the memory and outputs the logical faults addresses
4. Organize the outputs of the memory creator to simplify the checks of the memory reconstructor script

The previous steps were repeated to create a collection of memories and their related MEUs.

Now that the memories have been generated, a Python memory reconstructor script is called. This script performs the following operations:

1. Select a set of MEUs.
2. For each possible MDC and for each MEU in the set:
   (a) Reconstruct all the physical locations of the faults inside the MEU, using their logical address and assumed parameters.
   (b) Check the physical location to understand if all faults are adjacent to each other.
   (c) If the faults are all adjacent, add the current MDC to the possible correct ones.
3. Collect all the MDCs that were marked as "possibly correct" for all the MEUs in the set.

The output of the aforementioned steps is a list of memory parameters and the number of MEUs they can contain in the correct neighborhood as previously explained in Fig.3.63 and Fig.3.64.

Memories' and MEUs' parameters

The experiments were performed on around 5,500 different bit-scrambled (as shown in Fig.2.6) memories with randomly chosen MDCs presented in table.3.21. A minimum is decided in table3.21 because the number of the equivalents increases in small-size memories. Because there are few MDCs available, and most of them are equivalent geometrically. These small-size memories were not having a dramatic reduction in possible MDC.

For each memory, a set of 100 MEUs were injected. These MEUs were randomly generated with the following MDCs, also reported in Fig.3.62:

- Base Fault Count: Between 2 and 6
- Shapes: square, rectangle, single line (wordline or bitline oriented)
- Extra fault: a small probability exists that an extra fault is added adjacent to a previously inserted fault, such as the case in the bottom right of Fig.3.62

Given the aforementioned parameters, the algorithm tested a total of 550,000 MEUs distributed in 5,500 memories.

Stats and performances

The tests were performed on a Macbook Pro 13" equipped with an Apple M1 processor and 16GB of RAM. Table 3.22 summarizes our experimental results. It can be seen that each set of MEU had a different number of equivalent MDCs based on the number of considered MEUs (down to 2 for 100 MEUs). Notably, in the beginning, no data was available about the MDC. But among the candidate memories to be the correct memory with the correct parameters, the correct one is always present in this list. This cross-validation is based on the fact that the parameters were already known when random memories were generated.

Experimental results		
MEU sets per memory (#)	**Average equivalent MDCs (#)**	**Time**
20	6	26 minutes
40	3.64	45 minutes
60	2.79	63 minutes
80	2.31	77 minutes
100	2	96 minutes

Table 3.22: Experimental results and performances of the proposed algorithm.

Equivalent memories

For each MEU set, the algorithm correctly reconstructed the compliant memory parameters to what they belonged to, together with some "equivalent" memories. Fig. 3.66 and Fig.3.67 graphically explain why we have such equivalencies. For example, in Fig.3.66, it is possible to see how the same MEU made of four faults can come from memory with a straightforward organization (on top) or from memory with a Column mirror equal to 2 (on bottom). In Fig.3.67, another MEU of four faults can come from two different memories. One of the possible memories is mirrored in the center (top), and the other has a column mirror equal to 2 (bottom).

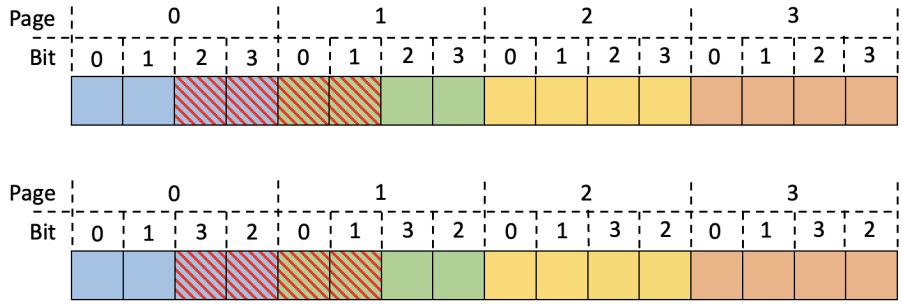


Figure 3.66: The MEU in the picture may come from a straightforward memory organization (top) or a mirroring every two columns (bottom)

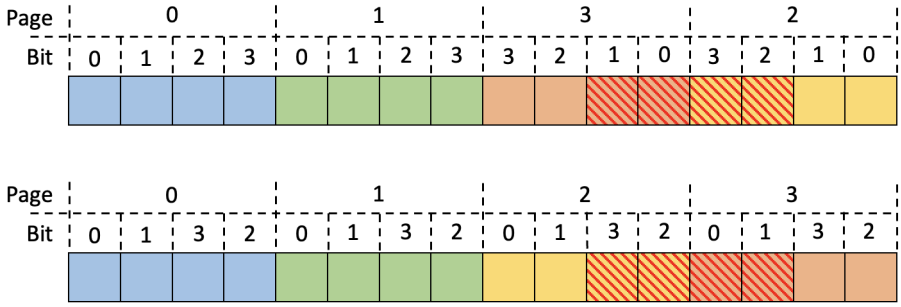


Figure 3.67: The MEU in the picture may come from an x-axe mirrored memory organization (top) or a mirroring every two columns one (bottom)

The main reason behind these equivalencies is that the location of two faults can be changed without violating the fault location shapes provided in Fig.3.62.

These examples show some of the possible equivalent memories organizations. The number of equivalent memories can decrease if the MEU has a greater dimension and is not symmetrical. In symmetrical shapes, all the faults can be swapped without violating the memory contiguity, and there are always memories with parameters that fit these MEU shapes.

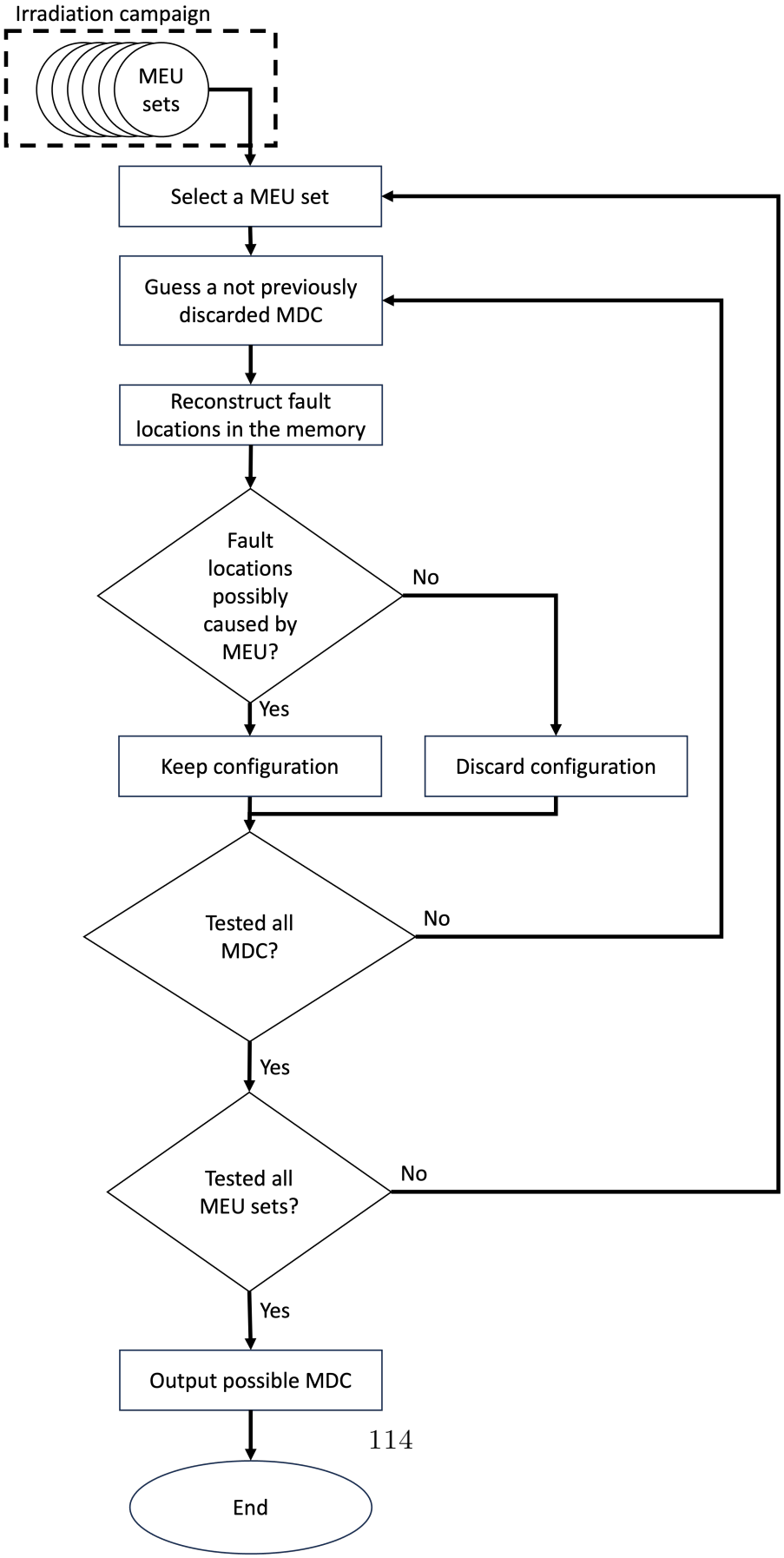


Figure 3.65: The flowchart of the proposed algorithm

Chapter 4

Conclusions and Future Work

This thesis has investigated several critical aspects of embedded memory testing and diagnosis in Automotive SoCs, spanning from manufacturing considerations to end-user reliability assessment. The contributions presented address the growing challenges of ensuring memory reliability in increasingly complex and safety-critical automotive systems.

A deep characterization of the memories embedded in automotive SoCs is presented, showing variation of key parameters in different conditions of supply voltage and access speed.

The development and validation of three novel on-chip diagnostic data collection and processing methods provide significant improvements in diagnostic efficiency. The slice-based method, achieving up to 99.8% diagnostic space savings compared with list-based representations, offers a lossless approach for failure bitmap encoding. The difference-based approach further improve this performance by 66.99% by only encoding the differences between one test step and the next one. The pixel slice method, while inherently lossy, achieves up to 72.6% savings and provides valuable density representation for rapid identification of weaker memory areas. The slice-based and the pixel methods were successfully implemented and validated on an Infineon Aurix SoC, demonstrating their practical applicability in a real-world automotive context.

Furthermore, the novel hardware-software scheme for diagnostic data retrieval demonstrated a substantial 25% reduction in test time while improving fault representation fairness, particularly beneficial in scenarios with limited on-chip memory. This contribution addresses the practical constraints of on-chip diagnostic memory resources and optimizes the utilization of available diagnostic space.

The thesis also addressed the challenge of voltage droop during testing, a crucial factor affecting data reliability, especially at minimum operating voltages. The two proposed low-area, low-complexity circuits provide effective mitigation strategies for voltage droop, ensuring reliable data collection for comprehensive failure analysis. These circuits offer practical solutions for minimizing voltage droop-induced failures

without significant overhead.

The development of a ResNet18-based neural network for fault classification introduced a novel approach capable of classifying eight distinct fault shapes. This method outperforms previous approaches and enables the recognition of novel fault shapes, contributing to a more comprehensive understanding of emerging failure mechanisms. The neural network's ability to identify new fault shapes is particularly valuable for adapting to evolving memory technologies and defect characteristics.

Finally, the software developed for extracting memory design configurations from irradiation test data empowers end-users to simulate radiation effects and assess memory reliability in their specific applications. This contribution bridges the gap between manufacturer knowledge and end-user needs, enabling more accurate reliability assessments in radiation-prone environments.

Future research directions include expanding the neural network's capabilities to classify a wider range of complex fault shapes. Further optimization of the diagnostic data compression algorithms could explore more sophisticated compression techniques while minimizing information loss. Extending the memory reconstruction software to accommodate diverse memory architectures would enhance its versatility and applicability across a broader spectrum of embedded memory systems. Additionally, exploring the integration of these techniques into a unified framework for comprehensive memory testing and diagnosis would provide a powerful tool for ensuring the reliability of embedded memories in future automotive SoCs. This future work will build upon the foundation laid by this thesis and contribute to the ongoing pursuit of robust and reliable embedded memory systems for the automotive industry.

[1] P. Bernardi et al. "Recent Trends and Perspectives on Defect-Oriented Testing". In: *2022 IEEE 28th International Symposium on On-Line Testing and Robust System Design (IOLTS)*. 2022, pp. 1–10. DOI: 10.1109/IOLTS56730.2022.9897647.

[2] A. L. Landzberg and R. Van Nostrand. *Microelectronics Manufacturing Diagnostics Handbook*. New York, USA, 1993.

[3] Wongi Hong, J. Choi, and H. Chang. "A Programmable Memory BIST for Embedded Memory". In: *Proceedings of the International SoC Design Conference*. Busan, Korea, Nov. 2008, pp. 24–25.

[4] C.H. Tsai and C.W. Wu. "Processor-programmable memory BIST for bus-connected embedded memories". In: *Proceedings of the Design Automation Conference*. Yokohama, Japan, Jan. 2001, 30–2 February.

[5] K. S. Das and P. Prakash. "Automatic MBIST Scheduling Engine". In: *Proceedings of the 2019 IEEE International Conference on Electronics, Computing and Communication Technologies (CONECCT)*. Bangalore, India: IEEE, 2019, pp. 1–6. DOI: 10.1109/CONECCT47791.2019.9012926.

[6] L. Degli Abbati et al. "Industrial best practice: cases of study by automotive chip- makers". In: *2021 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*. 2021, pp. 1–6. DOI: 10.1109/DFT52944.2021.9568350.

[7] Jiann-Jong Chen, Fong-Cheng Yang, and Chih-Chiang Chen. "A New Monolithic Fast-Response Buck Converter Using Spike-Reduction Current-Sensing Circuits". In: *IEEE Transactions on Industrial Electronics* 55.3 (2008), pp. 1101–1111. DOI: 10.1109/TIE.2007.907661.

[8] A. Manzini et al. "A Machine Learning-based Approach to Optimize Repair and Increase Yield of Embedded Flash Memories in Automotive Systems-on-Chip". In: *2019 IEEE European Test Symposium (ETS)*. 2019, pp. 1–6. DOI: 10.1109/ETS.2019.8791529.

[9] Jianbo Li et al. "A Hybrid Flow for Memory Failure Bitmap Classification". In: *2012 IEEE 21st Asian Test Symposium*. 2012, pp. 314–319. DOI: 10.1109/ATS.2012.16.

[10] N. Campanelli et al. "Cumulative embedded memory failure bitmap display & analysis". In: *13th IEEE Symposium on Design and Diagnostics of Electronic Circuits and Systems*. 2010, pp. 1–6. DOI: 10.1109/DDECS.2010.5654683.

[11] Alan D. Tipton et al. "Multiple-Bit Upset in 130 nm CMOS Technology". In: *IEEE Transactions on Nuclear Science* 53.6 (2006), pp. 3259–3264. DOI: 10.1109/TNS.2006.884789.

[12] Yujuan He et al. "Analysis of Atmospheric Neutron Radiation Effects in Automotive Electronics Systems". In: *2020 IEEE International Symposium on the Physical and Failure Analysis of Integrated Circuits (IPFA)*. 2020, pp. 1–4. DOI: 10.1109/IPFA49335.2020.9260970.

[13] A.J. Williams, M.R. McEwen, and A.R. DuSautoy. "Radiation testing for space applications at the National Physical Laboratory". In: *1998 IEEE Radiation Effects Data Workshop. NSREC 98. Workshop Record. Held in conjunction with IEEE Nuclear and Space Radiation Effects Conference (Cat. No.98TH8385)*. 1998, pp. 148–151. DOI: 10.1109/REDW.1998.731495.

[14] D. G. Mavis et al. "Multiple Bit Upsets and Error Mitigation in Ultra-Deep Submicron SRAMS". In: *IEEE Transactions on Nuclear Science* 55.6 (2008), pp. 3288–3294. DOI: 10.1109/TNS.2008.2006893.

[15] I. Schanstra et al. "Semiconductor Manufacturing Process Monitoring using Built-In Self-Test for Embedded Memories". In: *Proceedings International Test Conference 1998*. 1998.

[16] J.T. Chen et al. "Test response compression and bitmap encoding for embedded memories in manufacturing process monitoring". In: *Proceedings International Test Conference 2001 (Cat. No.01CH37260)*. 2001, pp. 258–267. DOI: 10.1109/TEST.2001.966641.

[17] P. Bernardi et al. "An efficient algorithm for the extraction of compressed diagnostic information from embedded memory cores". In: *2003 IEEE Conference on Emerging Technologies and Factory Automation*. 2003.

[18] R. Silveira, Q. Qureshi, and R. Zeli. "Flexible architecture of memory BISTs". In: *Proceedings of the 2018 IEEE 19th Latin-American Test Symposium (LATS)*. Sao Paulo, Brazil, Mar. 2018, pp. 1–6. DOI: 10.1109/LATW.2018.8349666. URL: https://doi.org/10.1109/LATW.2018.8349666.

[19] S.B. Ghale and P. Namita. "Design and Implementation of Memory BIST for Hybrid Cache Architecture". In: *Proceedings of the 2021 6th International Conference on Communication and Electronics Systems (ICCES)*. Coimbatre, India, July 2021, pp. 26–31. DOI: 10.1109/ICCES51350.2021.9489225. URL: https://doi.org/10.1109/ICCES51350.2021.9489225.

[20] R. Zeli, R. Silveira, and Q. Qureshi. "SoC Memory Test Optimization using NXP MTR Solutions". In: *Proceedings of the 2019 IEEE Latin American Test Symposium (LATS)*. Santiago, Chile, Mar. 2019, pp. 1–5. DOI: 10.1109/LATW.2019.8704566. URL: https://doi.org/10.1109/LATW.2019.8704566.

[21] A. Manzone et al. “Integrating BIST techniques for on-line SoC testing”. In: *Proceedings of the 11th IEEE International On-Line Testing Symposium*. French Riviera, France, July 2005, pp. 235–240. DOI: 10.1109/IOLTS.2005.38. URL: https://doi.org/10.1109/IOLTS.2005.38.

[22] Paolo Bernardi and Lyl Ciganda. “An Adaptive Low-Cost Tester Architecture Supporting Embedded Memory Volume Diagnosis”. In: *IEEE Transactions on Instrumentation and Measurement* 61.4 (2012), pp. 1002–1018. DOI: 10.1109/TIM.2011.2179822.

[23] T. McLaurin, F. Frederick, and R. Slobodnik. “The DFT challenges and solutions for the ARM® Cortex™-A15 Microprocessor”. In: *Proceedings of the 2012 IEEE International Test Conference*. Anaheim, CA, USA, Nov. 2012, pp. 1–9. DOI: 10.1109/TEST.2012.6401534. URL: https://doi.org/10.1109/TEST.2012.6401534.

[24] P. Bernardi et al. “Optimized diagnostic strategy for embedded memories of Automotive Systems-on-Chip”. In: *2022 IEEE European Test Symposium (ETS)*. 2022, pp. 1–6. DOI: 10.1109/ETS54262.2022.9810445.

[25] G. Insinga et al. “Density-oriented diagnostic data compression strategy for characterization of embedded memories in Automotive Systems-on-Chip”. In: *2023 IEEE European Test Symposium (ETS)*. 2023, pp. 1–6. DOI: 10.1109/ETS56758.2023.10174126.

[26] Li-C. Wang. “An Autonomous System View To Apply Machine Learning”. In: *2018 IEEE International Test Conference (ITC)*. 2018, pp. 1–10. DOI: 10.1109/TEST.2018.8624844.

[27] Li-C. Wang. “Experience of Data Analytics in EDA and Test—Principles, Promises, and Challenges”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 36.6 (2017), pp. 885–898. DOI: 10.1109/TCAD.2016.2621883.

[28] Cristian Zambelli, Rino Micheloni, and P. Olivo. “Machine Learning for 3D NAND Flash and Solid State Drives Reliability/Performance Optimization”. In: *Machine Learning and Non-volatile Memories*. Ed. by Rino Micheloni and Cristian Zambelli. Cham: Springer International Publishing, 2022, pp. 133–156. ISBN: 978-3-031-03841-9. DOI: 10.1007/978-3-031-03841-9_7. URL: https://doi.org/10.1007/978-3-031-03841-9_7.

[29] Debao Wei et al. “Research on prediction model for NAND flash bit errors”. In: *2015 12th IEEE International Conference on Electronic Measurement and Instruments (ICEMI)*. Vol. 01. 2015, pp. 233–238. DOI: 10.1109/ICEMI.2015.7494259.

[30] Haichun Zhang et al. "An SVM-Based NAND Flash Endurance Prediction Method". In: *Micromachines* 12.7 (2021). ISSN: 2072-666X. DOI: 10.3390/mi12070746.

[31] Chuanhe Shan et al. "Systematic defect detection methodology for volume diagnosis: A data mining perspective". In: *2017 IEEE International Test Conference (ITC)*. 2017, pp. 1–10. DOI: 10.1109/TEST.2017.8242050.

[32] Matthew Nero et al. "Concept Recognition in Production Yield Data Analytics". In: *2018 IEEE International Test Conference (ITC)*. 2018, pp. 1–10. DOI: 10.1109/TEST.2018.8624714.

[33] Chuanhe Shan et al. "Deploying A Machine Learning Solution As A Surrogate". In: *2019 IEEE International Test Conference (ITC)*. 2019, pp. 1–10. DOI: 10.1109/ITC44170.2019.9000109.

[34] Luzhou Peng, Bowen Qiang, and Jiacheng Wu. "A Survey: Image Classification Models Based on Convolutional Neural Networks". In: *2022 14th International Conference on Computer Research and Development (ICCRD)*. 2022, pp. 291–298. DOI: 10.1109/ICCRD54409.2022.9730565.

[35] Ayodeji Olalekan Salau and Shruti Jain. "Feature Extraction: A Survey of the Types, Techniques, Applications". In: *2019 International Conference on Signal Processing and Communication (ICSC)*. 2019, pp. 158–164. DOI: 10.1109/ICSC45622.2019.8938371.

[36] L. Amoroso et al. "Single shot transient suppressor (SSTS) for high current high slew rate microprocessor". In: *APEC '99. Fourteenth Annual Applied Power Electronics Conference and Exposition. 1999 Conference Proceedings (Cat. No.99CH36285)*. Vol. 1. 1999, 284–288 vol.1. DOI: 10.1109/APEC.1999.749649.

[37] A. Barrado et al. "Output filter influence on the fast response double buck DC-DC converter (FRDB)". In: *2004 IEEE 35th Annual Power Electronics Specialists Conference (IEEE Cat. No.04CH37551)*. Vol. 6. 2004, 4696–4701 Vol.6. DOI: 10.1109/PESC.2004.1354829.

[38] V. Boscaino et al. "Non-linear digital control improving transient response: Design and test on a multiphase VRM". In: *SPEEDAM 2010*. 2010, pp. 507–512. DOI: 10.1109/SPEEDAM.2010.5542037.

[39] Sivaraman Masilamani et al. "A High-Speed, Non-Linear Control Based Voltage Droop Mitigation Technique for Integrated Voltage Regulators in Modern Microprocessors". In: *2019 IEEE Applied Power Electronics Conference and Exposition (APEC)*. 2019, pp. 2241–2246. DOI: 10.1109/APEC.2019.8722207.

[40] Wongi Hong, J. Choi, and H. Chang. "A programmable memory BIST for embedded memory". In: *2008 International SoC Design Conference*. Vol. 02. 2008, pp. II-195-II–198. DOI: 10.1109/SOCDC.2008.4815717.

[41] Ching-Hong Tsai and Cheng-Wen Wu. "Processor-programmable memory BIST for bus-connected embedded memories". In: *Proceedings of the ASP-DAC 2001. Asia and South Pacific Design Automation Conference 2001 (Cat. No.01EX455)*. 2001, pp. 325–330. DOI: 10.1109/ASPDAC.2001.913327.

[42] Paolo Bernardi et al. "Built-In Self-Test Architecture Enabling Diagnosis for Massive Embedded Memory Banks in Large SoCs". In: *Electronics* 13.2 (2024). DOI: 10.3390/electronics13020303. URL: https://www.mdpi.com/2079-9292/13/2/303.

[43] Abhijit Bendale and Terrance E. Boult. "Towards Open Set Deep Networks". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 1563–1572. DOI: 10.1109/CVPR.2016.173.

[44] Radosvet Desislavov, Fernando Martínez-Plumed, and José Hernández-Orallo. "Trends in AI inference energy consumption: Beyond the performance-vs-parameter laws of deep learning". In: *Sustainable Computing: Informatics and Systems* 38 (2023), p. 100857. ISSN: 2210-5379. DOI: https://doi.org/10.1016/j.suscom.2023.100857. URL: https://www.sciencedirect.com/science/article/pii/S2210537923000124.

[45] K. Nanthini et al. "A Survey on Data Augmentation Techniques". In: *2023 7th International Conference on Computing Methodologies and Communication (ICCMC)*. 2023, pp. 913–920. DOI: 10.1109/ICCMC56507.2023.10084010.

[46] Sebastien C. Wong et al. "Understanding Data Augmentation for Classification: When to Warp?" In: *2016 International Conference on Digital Image Computing: Techniques and Applications (DICTA)*. 2016, pp. 1–6. DOI: 10.1109/DICTA.2016.7797091.

[47] Agnieszka Mikołajczyk and Michał Grochowski. "Data augmentation for improving deep learning in image classification problem". In: *2018 International Interdisciplinary PhD Workshop (IIPhDW)*. 2018, pp. 117–122. DOI: 10.1109/IIPHDW.2018.8388338.

[48] Saikiran Bulusu et al. "Anomalous Example Detection in Deep Learning: A Survey". In: *IEEE Access* 8 (2020), pp. 132330–132347. DOI: 10.1109/ACCESS.2020.3010274.

[49] Christoph Berger et al. "Confidence-Based Out-of-Distribution Detection: A Comparative Study and Analysis". In: *Uncertainty for Safe Utilization of Machine Learning in Medical Imaging, and Perinatal Imaging, Placental and Preterm Image Analysis*. Ed. by Carole H. Sudre et al. Cham: Springer International Publishing, 2021, pp. 122–132. ISBN: 978-3-030-87735-4.

[50] Dan Hendrycks and Kevin Gimpel. *A Baseline for Detecting Misclassified and Out-of-Distribution Examples in Neural Networks*. 2018. arXiv: 1610.02136 [cs.NE].

[51] Akshay Raj Dhamija, Manuel Günther, and Terrance E. Boult. "Reducing network agnostophobia". In: *Proceedings of the 32nd International Conference on Neural Information Processing Systems*. NIPS'18. Montréal, Canada: Curran Associates Inc., 2018, pp. 9175–9186.

[52] Elena Limonova et al. "ResNet-like Architecture with Low Hardware Requirements". In: *2020 25th International Conference on Pattern Recognition (ICPR)*. 2021, pp. 6204–6211. DOI: 10.1109/ICPR48806.2021.9413186.

[53] Gaurav Kumar Pandey and Sumit Srivastava. "ResNet-18 comparative analysis of various activation functions for image classification". In: *2023 International Conference on Inventive Computation Technologies (ICICT)*. 2023, pp. 595–601. DOI: 10.1109/ICICT57646.2023.10134464.

[54] Olga Russakovsky et al. "ImageNet Large Scale Visual Recognition Challenge". In: *Int. J. Comput. Vision* 115.3 (Dec. 2015), pp. 211–252. ISSN: 0920-5691. DOI: 10.1007/s11263-015-0816-y. URL: https://doi.org/10.1007/s11263-015-0816-y.

[55] NPlusT. *BarnieMAT*. https://www.n-plus-t.com/products/barniemat/. 2024.

[56] P. Bernardi, G. Insinga, and N. Kolahimahmoudi. "A Novel Approach to Extract Embedded Memory Design Parameter Through Irradiation Test". In: *2023 IFIP/IEEE 31st International Conference on Very Large Scale Integration (VLSI-SoC)*. 2023, pp. 1–6. DOI: 10.1109/VLSI-SoC57769.2023.10321848.

This Ph.D. thesis has been typeset by means of the TEX-system facilities. The typesetting engine was pdfLATEX. The document class was `toptesi`, by Claudio Beccari, with option `tipotesi=scudo`. This class is available in every up-to-date and complete TEX-system installation.