Optimized Embedded Memory Diagnosis

M. de Carvalho, P. Bernardi, M. Sonza Reorda
Politecnico di Torino
Dip. di Automatica e Informatica
Torino, Italy

N. Campanelli, T. Kerekes
NplusT
Semiconductor Application Center
Montecastrilli (TR), Italy

D. Appello, M. Barone, V. Tancorre, M. Terzi
STMicroelectronics
Agrate Brianza (MI), Italy

***Abstract*—This paper describes an optimized embedded memory diagnosis flow that exploits many levels of knowledge to produce accurate failure hypothesis. The proposed post-processing analysis flow is composed of many steps investigating failure shapes as well as cell fail syndromes, and includes advanced techniques to tackle incomplete data possibly due to tester noise and/or by faults showing intermittent effects. The effectiveness of the technique is demonstrated on an automotive-oriented System-on-Chip (SoC) manufactured in a 90nm technology by STMicroelectronics, which includes embedded SRAM memory cores tested using a programmable BIST. Scrambled BITMAPS gives a visual feedback leading to quick physical defect identification. Such research is relevant to aid on the manufacturing, material and process enhancements raising silicon yield.**

Keywords: ***Embedded memory diagnosis; Fault models; Scrambling; Failure Bitmaps; Failure analysis***

I. INTRODUCTION

Silicon debug and diagnosis processes are one of the most challenging tasks in the semiconductor industries, especially for newly designed SoC architectures. Effective diagnosis can be achieved via the combination of many techniques including insertion of on-chip design for testability hardware modules, test equipments applying suitable circuit stimulation, and software analysis tools to post process the retrieved data.

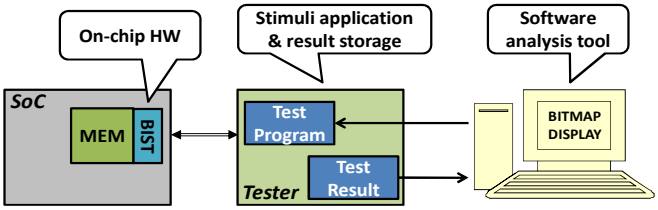


Figure 1: Memory diagnosis flow components

In the SoC arena, the test and diagnosis of embedded memory cores is usually based on Built-In-Self-Test (BIST)[1][2] modules, especially optimized to quickly achieve accurate and reliable results. Since the diagnosis of failing memory cores is one of the major issues, BIST solutions offering diagnosis features (DBIST [3][4]) are popular, and permit to extract from the silicon a comprehensible and accurate report including the unexpected test responses (called *failure syndrome*). DBIST modules are often designed to implement various testing procedures, e.g., activating several test algorithms or special walking modes.

The BIST procedure is managed by automatic test equipment (ATE) or tester which is responsible for launching and monitoring the test algorithms applied to the memories. The ATE has to guarantee flexibility and must have enough intelligence to produce stimulations that may change depending on the encountered faults. Results stored in the ATE memory are finally sent out in the form of different files (eventually encapsulated into standard frames such as dictated by the Motorola's SRECORD and Standard Test Data Format - STDF standard formats) to be further analyzed.

Software analysis tools primary outcome is to pinpoint failure modes and to indicate probable fault locations of failing chips. This is crucial to speed-up the failure analysis process which gives precious feedback to memory designers in a closed loop. To perform such a post-processing analysis, software tools have to put together all the test environment information, encompassing embedded memories characteristics, such as scrambling parameters, BIST features, applied test algorithms, and eventual limitation due for instance to the available ATE memory.

This paper describes an effective technique to obtain meaningful information about location and type of embedded memory physical defects. The methodology is based on a visualization technique for failure bitmaps [5], whose output is augmented by

- analyzing failure shapes to restrict physical research efforts, since many logical failures are due to a single defect
- putting them in relation with fault models, joined with a confidence score.

Furthermore, the proposed technique tries to cope with noise in the testing environment, often observed when performing diagnosis at high temperatures, and intermittent failing behaviors. This is a very serious problem that to the best of our knowledge has never been mentioned before in the literature on memory diagnosis. In addition, the approach is relevant to understand manufacture, material and process errors unperceived by human verification, thus increasing the yield in production.

Examples and results reported in the following paragraphs are related to SRAM memories embedded into a 90nm automotive SoC family by STMicroelectronics. Results gathered on such an industrial case study demonstrate the effectiveness of the approach in a real scenario.

978-1-4244-9756-0/11/$26.00 ©2011 IEEE

The rest of the paper is structured as follows: Section 2 introduces the basic concepts about embedded memory diagnosis, by describing first memory topology, and then fault models, BIST concepts and possible test framework; Section 3 outlines the proposed additional steps in the diagnosis process; Section 4 shows the results obtained. Finally, conclusions are drawn in Section 5.

II. EMBEDDED MEMORY DIAGNOSIS BACKGROUND

This section introduces some important concepts in the area of memory diagnosis.

A. Memory topology

The first outcome for memory analysis software tool is to represent the memory under analysis as it is physically structured and positioned. For this reason, the logical vision (usually appearing as a very long cell array whose width corresponds to the word parallelism) has to be translated into the physical one, which is an optimized map whose dimension is determined by a set of parameters constituting the so-called **scrambling** schema. [5]

Other than the basic information such as the number of memory words (WORDS value) and the number of bits per word (BITS value), a scrambling schema is usually characterized by:

- MUX value: indicates the number of logical words included in a single physical row
- BIT ORDER sequence: describes how the bits of the logical words are organized in a physical row
- MIRROR values: whereas a memory presents a regular structure, memory portions can be horizontally and/or vertically repeated, eventually with an alternate order.

Example - In a typical embedded SRAM scenario, in case the MUX value is N, every physical row is composed of BITS blocks, each one including N bits; block i commonly includes all the i^{th} bits of the N words composing the row. Furthermore, bits in a block are normally organized according to the word address. Figure 2 illustrates this basic BIT ORDER sequence information.

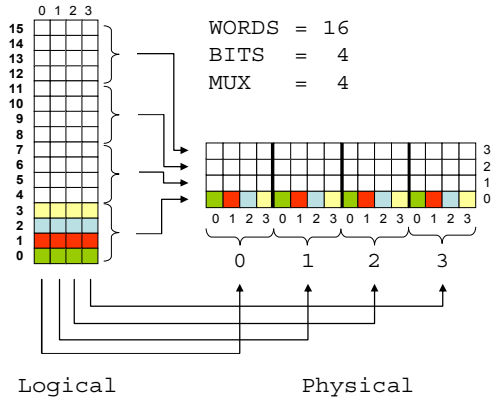


Figure 2: basic mapping between logical and physical SRAM view

To have this view of the memory array is crucial to succeed in the physical failure inspection since it eases the microscope alignment process. It is extremely important to map each single physical cell in order to apply the correct failure recognition algorithm thus providing better diagnosis results.

B. Bitmap display tool working principle

As briefly described in the introductive section, a complete and effective flow to investigate embedded memory failures has to be performed using test features at several levels. In the final one, which is aimed at post-processing the failures discovered by the on-chip BIST circuitry under the support of the ATE, a software analysis tool should feedback hints about physical failure modes. In other word, such software is expected to provide a logical interpretation and possibly lead to an accurate fault hypothesis. Failure analysis is strongly simplified in the case the analysis returns topological information and hypothesis about the failing mode; using this kind of information, the failure analysts can quickly provide feedback to designers for technology and design tuning.

Figure 3 shows the conceptual view for a memory analysis bitmap display tool.

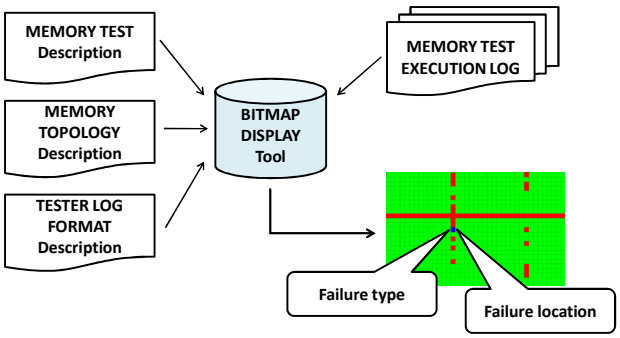


Figure 3: conceptual view of a software analysis tool able to draw memory failure bitmaps

To be easily readable, final results may be shown in a graphical format, usually the so-called *memory failure bitmap*. This format represents the best way to drive failure analysis efforts, since it permits to have a preliminary faulty mask to observe when performing physical inspections.

Anyway, since the shape alone is not necessarily embedding all the key information for correctly classifying the results, some additional failure mode information is really desirable.

C. Fault models

Fault models are engineering models of a failing cell within a memory array that could manifest failure and is unable to function as specified. The models are used to predict consequences and localization of a particular fault, allowing improving fabrication process accuracy [6].

The most used fault models are:

1. *Stuck at 0 (SA0)*: the cell's content is always 0, even if a write 1 has been performed.
2. *Stuck at 1(SA1)*: the cell's content is always 1, even if a write 0 has been performed.
3. *Transition ↑ / 0 (TF1)*: the cell fails to undergo a 0 → 1 transition.
4. *Transition ↓ / 1 (TF0)*: the cell fails to undergo a 1→ 0 transition.
5. *Inversion Coupling (CFin)*: a transition in one cell inverts the contents of a second cell.
6. *Idempotent Coupling (CFid)*: a transition in one cell forces the contents of a second cell to a certain value (0 or 1).
7. *Bridging (BF)*: this fault model involves any number of cells and is caused by a logic level rather than a transition. It normally consists of a galvanic connection

between two (or more) cells, or lines and a state of a line.

8. *State Coupling (SCF)*: a coupled cell or line is forced to a certain value, when the coupling cell or line is in a given state.
9. *Active Neighborhood Pattern Sensitive (ANPSF)*: the content of a cell, or the ability to change its contents, is influenced by the contents of other cells in the memory. ANPSF corresponds to the base cell changing its contents due to a change in the values hold by the neighborhood cells.
10. *Passive Neighborhood Pattern Sensitive (PNPSF)*: the content of the base cell cannot be changed (i.e., it cannot make a transition) due to a certain neighborhood pattern.

This set of classical fault models can be extended including technology-oriented fault models, and may consider also intermittent fault effects. In fact, it is quite common to observe incomplete fault effects; as an example, it is sometimes observed that, even in the case of faults identified as SA faults using optical inspection methods, the returned syndromes is not perfectly matching with the expected syndrome for this kind of fault.

The most used strategy to assign a faulty behavior to a fault model bases on dictionaries, which store the set of possible syndromes that a faulty memory core can produce. To perform a correlation between the retrieved syndrome and the fault model causing it, the dictionary is accessed to select the most probable failure type hypothesis.

III. THE OPTIMIZED DIAGNOSIS FLOW

In this paper we propose to extend the usual memory diagnosis methods by using different sets of failure information. In particular, to adopt a software environment able to predict the single and group failing behavior of all the relevant cells in the memory.

The optimized diagnosis flow results on the display an augmented failure bitmap which is built by performing three logic analysis steps with the results returned by the ATE:

1) Failure shape recognition over the memory array, with data correction

2) Single failing cell fault model recognition through fault dictionaries

3) Final fault hypothesis is obtained by mapping and eventually correcting single cell information over failing shapes.

The flow in figure 4 is essential for an effective diagnosis, especially because of test inconsistency that may affect test procedures due to the following factors

- while the automatic test equipment is working, probable field stresses can introduce noise causing erroneous data registration and possibly resulting in false positives values
- datalog truncation, that is shortening the test, may be done by engineers to reduce overall diagnosis time on the tester
- intermittent fault behavior may hide the real nature of the fault that could finally become permanent when inspection is performed at high temperature or after the burn-in process.

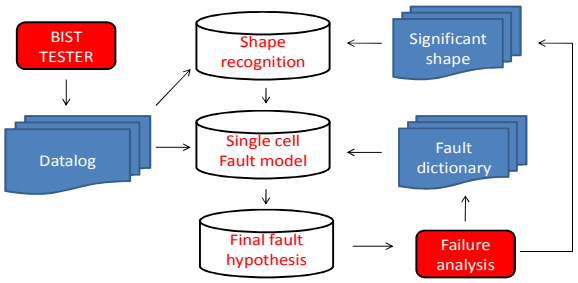


Figure 4: Diagnosis flow

The final result obtained by the software analysis tool which implements the aforementioned computation is the identification of failing memory areas joined with a preliminary fault type guess that is suitable to drive the efforts during the following failure analysis. In this way we can feedback more than a single piece of information about the location, but we can also try to understand the failing mode caused by the physical defect. This is particularly important to quickly close the loop within testing, failure analysis and design.

The next paragraphs detail the components of the proposed technique including some significant examples.

A. *Failure shape recognition and completion*

The initial step corresponds to identifying failures occurring in the memory cells, independently of their nature and organizing them into shapes (e.g., failing row, failing column, failing cluster).

In this phase, it does not matter the number of times the cell has shown a failing behavior along the test algorithm execution, but just if it can be included into a particular shape taking also into account other faults appearing in the memory array. Typically, the list of interesting shapes includes:

- Columns and rows
  - complete
  - partial
  - jeopardized
- Pairs
- Clusters
  - Regular
  - Irregular
- Spots.

Additionally, any regular failing schema in the memory array may be considered as caused by a single physical defect.

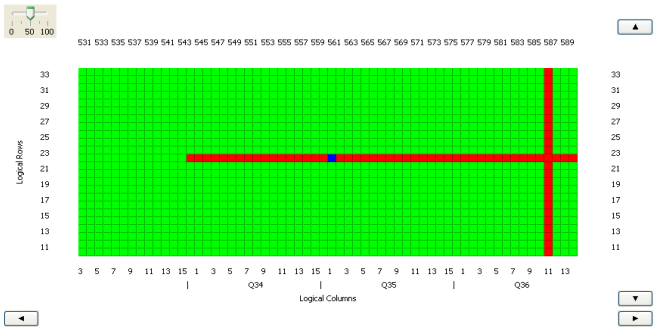


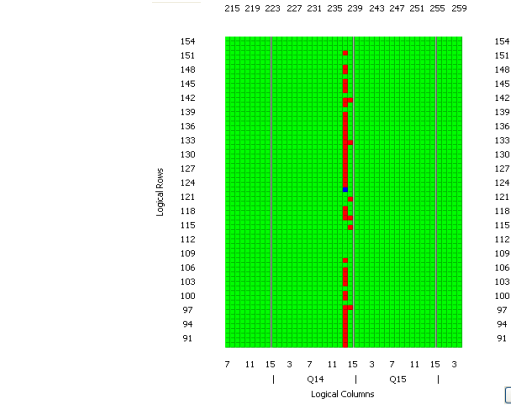
Figure 5: Bitmap with a partial row and a column shaped failures

In figure 5, a failure bitmap chunk that includes a couple of significant failing shapes is shown. One is a partial row failure, which starts close to the right side of the embedded memory and involves a relatively low number of cells. The starting location is the first candidate to investigate. Anyway, it is

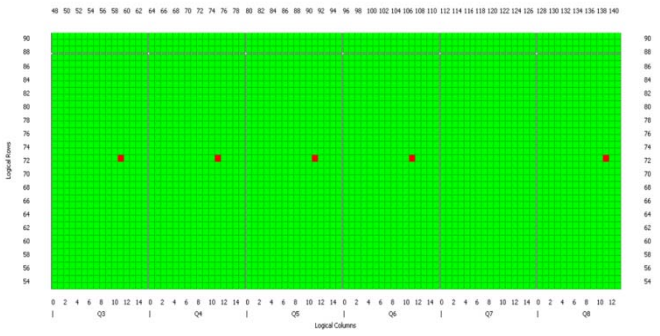
useful to go through a fault model investigation in order to obtain more reasons to attack this side of the failure.

The second is a complete column; this case is more difficult and not enough information is returned at this knowledge level. Furthermore, the failure shape is quite large since it involves all the cells from top to bottom of the memory array. This is not a problem at this level, but may become a factor to take into consideration when a fault model hypothesis is formulated.

It is sometimes useful to apply some corrections to the received data. As stated in section III, noise may interfere in the test flow, causing misleading and often random unexpected behaviors. At the shape recognition level, the possibility exists that some cells which are expected to fail in a regular failing sequence are not detected as faulty. This is a suspect behavior and may sometimes be corrected. Figure 6.a reports a column shaped defect whose datalog was corrupted at the BIST or ATE level.



(a)



(b)

Figure 6: (a) failure bitmap corrupted by noise and (b) intermittent fault effect.

Another unexpected behavior that could manifest itself during memory diagnosis is the intermittent fault effect. In many cases, a regular and clean sequence of failing cells is showing some holes in the sequence. Figure 6.b shows a case of jeopardized row where an expected to fail cell is not identified as faulty by the test procedure.

B. Single cell fault model recognition

The second step of the proposed test result analysis is intended to assign a fault model to each cell in an identified failure shape. This activity is crucial to reduce the manual efforts during physical inspections, since it returns indications about what kind of defect to search.

The fault model recognition step is based on accessing a dictionary including the syndromes of the most common fault models. The responses of failing cells in a shape are compared with known fault effects to produce a preliminary and completely logical hypothesis about the physical defect. This

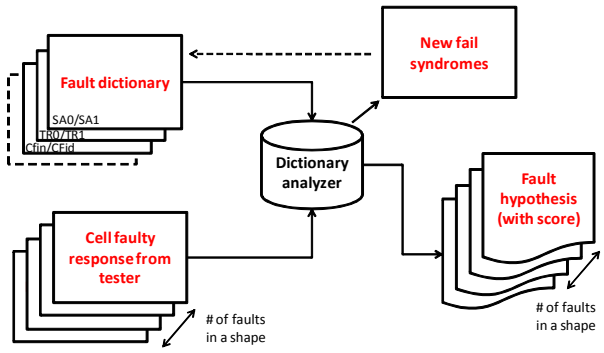


Figure 7: logical hypothesis produced by comparing the gathered syndrome with known fault type responses.

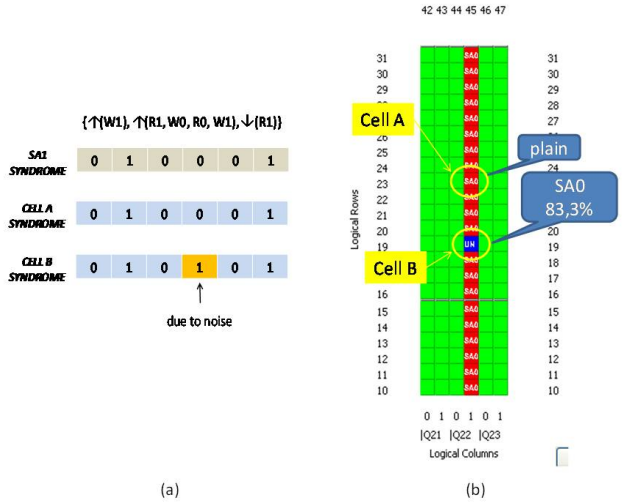
It is quite usual to have unknown responses with respect to the traditional fault models. For this reason, it makes sense to add new fail syndromes when it is verified that it is produced by a well isolated physical defect.

This step appears to be trivial, but it is not. In this scenario, at least 2 problematic situations may appear:

- Information is affected by noise, like in the shape recognition phase
- Datalog is truncated because test was stopped before retrieving all failure information.

The former situation may slightly deviate the analysis from the right diagnosis. To cope with this undesired effect, in this phase a score is given, corresponding to the matching percentage of the cell response with respect to a fault dictionary. As described in the next paragraph, this information will be completed and confirmed/changed in the final step of the technique.

Figure 8 illustrates an example of noisy context. In this particular case, the memory includes a failing cluster which is identified to be almost completely composed of cells affected by SA0 faults. Anyway, some cells show a haltered behavior with respect to the responses that are not belonging to the expected SA0 syndrome.



(a) (b)

Figure 8: noisy test environment affecting fault model selection; in (a) the March test executed and a comparison within expected and obtained syndromes, in (b) the shape of the failure including fault model hypothesis.

In this explanatory scenario, cells showing a behavior like cell B are labeled as SA1 with confidence of the 83.3% since there are 5 dictionary values out of 6 that are matching.

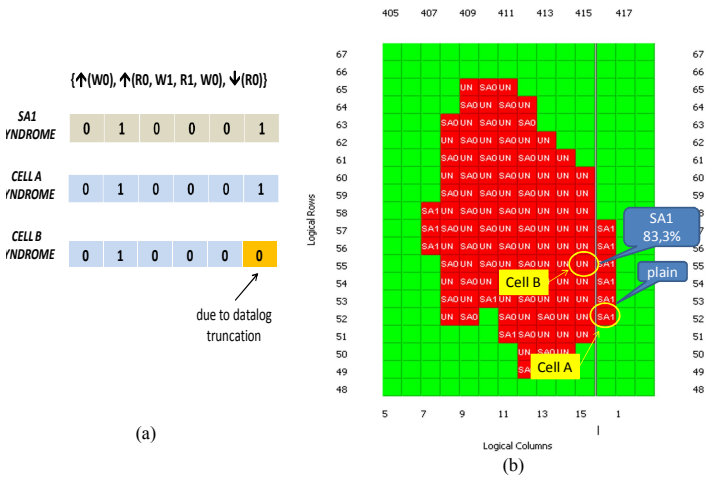


Figure 9: effect of data truncation; in (a) the March test executed and a comparison within expected and obtained syndromes, in (b) the shape of the failure including fault model hypothesis.

Concerning the data truncation, it usually intervenes when the physical defect affects a large set of cells failing many times; a typical scenario for truncation is the case of an entire column or a large cluster failing in a SA manner. In this unfortunate situation, the tester and/or the BIST impose a maximum number of fails to be recorded. As a consequence, it is possible that the result retrieval is incomplete. Figure 9 illustrates a possible context affected by data truncation; the diagnostic data retrieval is stopped in the middle of the column after the maximum number of failing information is reached, thus leading to different categorization of failures belonging to different parts of the defective area. In fact, we notice a set of plain SA1 and a set of SA1 with a 83.3% score (5 out of 6 values are matching)

Similarly to noise and data truncation effects, also intermittent faults are possibly haltering the test results. As well, they are tackled by assigning a confidence score which is then taken into consideration in the final step.

C. *Final fault hypothesis*

In the final fault hypothesis step, all the information related to a shape are put together to provide a final useful logical identification of a step. The following sequence of data manipulation operations are performed

1. all faults incompletely identified are possibly assigned to a plain fault model; this operation is done whereas the given score is higher than a given threshold (80% is usually a reasonable value)
2. missing data in the shapes are filled, reporting a plain fault model indication whereas the rest of the shape is providing a regular failing schema.

Let us consider the examples given in figure 6.a, 6.b, 8 and 9. They were finally assigned to the following failure categories, respectively:

- 6.a: SA0 full failing column
- 6.b: Address fault affecting all bit13 cells in a row
- 8: SA0 full failing column
- 9: SA1/SA0 cluster.

The user is only observing the final results, therefore it is not requested to do any manual activity to isolate the fault. Furthermore, with the frequent usage of the tool, new significant shapes and fault models can be added to the software environment to refine the diagnostic results.

At this point, failure analysis can proceed. As demonstrated by the cases of study included in section IV, the approach is suitable to perform a fast identification of the likely causes for a failure.

IV. CASES OF STUDY

Our cases of study consist of embedded SRAM memories of a 90nm SoC family developed by STMicroelectronics. Nonetheless, the approach can be applied to other non-embedded memories by modifying their tester to retrieve a valid SRECORD format.

These cores are tested with the support of a diagnostic BIST which can be programmed with an appropriate microcode describing march-like and non-march test algorithms. Usually the BIST is asked to reproduce a 36 elements memory test using several 2 data background values and 4 walking combination (solid, checkerboard, row and column stripes).

The BIST functionalities are activated both at wafer sort level measuring at several temperature, and at packaged level by using the Teradyne J750 tester. The data logging capability is actually limited to 500 failure responses; the results are dump to SRECORD standard compliant files to be further analyzed. In addition, the complexity of diagnosing large memories lies on the tester's memory size, whether it can store a large quantity of information or not. Otherwise, the test must be applied into small arrays of the embedded memory so that data are retrieved in blocks and diagnosis performed after all parts are put together.

The software analysis tool implementing the methodology described in section III having the functionalities needed to graphically display failure bitmaps, is a java application providing many hints to isolate physical defects and to evaluate their incidence on the entire population of failing chips, such as thorough cumulative analysis. It is implemented and released by NplusT Semiconductor Application Center and Politecnico di Torino.

In the following subsections, we report 4 cases of successful failure analysis driven by a preliminary logical analysis performed using the devised software tool. They are the most accurate diagnosis concurring with its physical analysis provided by STMicroelectronics and are described in the different following cases. In total, 500 different memories were diagnosed, and their cells had an statistics of 73% stuck-at faults, 22% unknown faults, 2% address faults, and 3% transitions, coupling and neighborhood sensitive faults.

Case 1

In this case, the applied sequence of march elements allowed isolating a failure whose shape is a pair involving two neighboring cells. Figure 10 is showing the logical and the physical view of the failure.

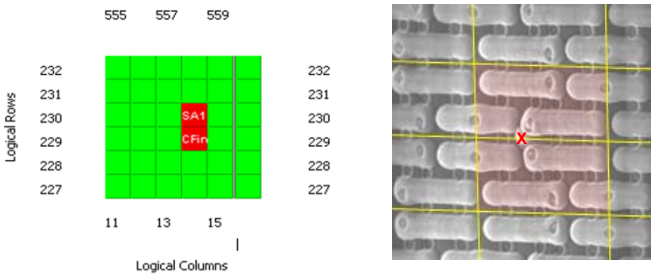


Figure 10. Pair defect: logical and physical appearance of the faulty array

According to the illustrated flow without any correction of the identified fault model, the top cell was modeled by the tool as a Stuck-at-1, while its neighbor as a coupling inversion fault. The photography of the physical memory structure was taken by a microscope and an unexpected impure element was observed which is causing the mentioned faulty effect. The top cell, which is always at high logic level, also acts as an aggressor cell to the lower victim one that suffers a coupling inversion fault.

Case 2

In this second experimental case study, an irregular shape was considered. It is composed of 2 SA1 faults (syndrome is perfectly matching the expected SA1 model), a SA0 (corrected after observing a 94% score) and a SA1 (corrected after 86% score) which is highlighted in blue.

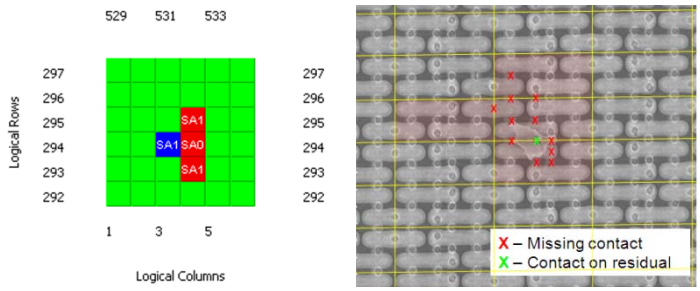


Figure 11. An irregular cluster showing many physical defects.

Case 3

The case shown in figure 12 reports a regular cluster shape defect. It's verified that the cluster is not fully modeled as a SA1, but it also include a SA0 fault. This hint was used to better investigate the behavior of this particular cell which is showing an interrupted contact more than other cells in the cluster.

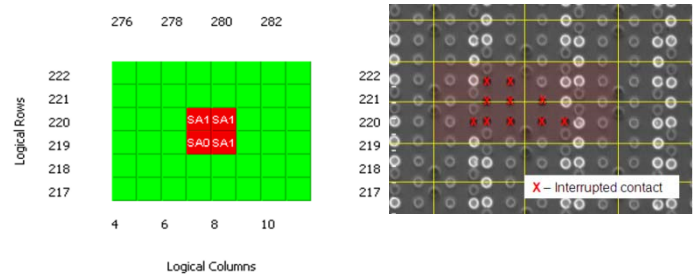


Figure 12. A regular cluster showing a composite logical hypothesis.

From the physical defect inspection point of view, the failing zone was quickly identified. The various nature of faults, logically pinpointed by the software analysis tool, permitted to isolate the whole set of manufacturing imperfections. It can be seen a macro defect in the middle of the cell group which is material residual producing an unwanted contact, but also some missing contacts were "hidden" beyond the major impurity. For this second group of failures, their investigation was dictated by the response of the tool, while a manual analysis could even have missed them.

Case 4

The case in figure 13 reports an irregular cluster shape defect, including both SA0 and SA1 behaviors. The physical defect observed during the failure analysis highlighted a large defect involving many paths to active area.

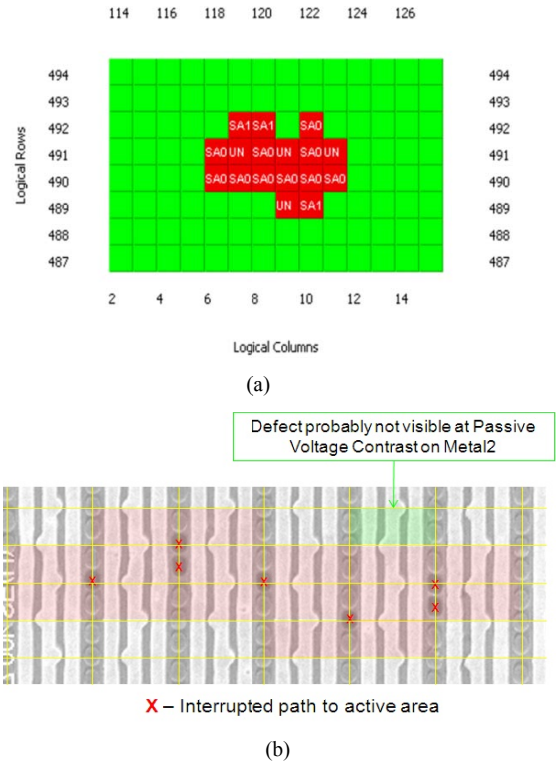


Figure 13. An irregular cluster showing a composite logical hypothesis.

One of the most interesting reasoning that can be done on this picture is related to the cell at coordinate 492/122. Apparently, this cell is not affected by any defect when observed using a passive voltage contrast technique and could be lost if not pinpointed by the software analysis.

V. CONCLUSIONS AND FUTURE WORKS

A software framework suitable to provide accurate logical fault hypothesis that are key to a quick failure analysis process has been proposed. The proposed methodology and the software analysis tool implementing it is able to categorize the failing results stemming from a memory test and to provide both information about the physical location of the defect and, possibly, a preliminary explanation of the corruption performed on the memory array. The included cases of study demonstrate the effectiveness of the approach on real case studies.

As future work, we are planning to extend the capability of the approach to associate each failure with a physical defect hypothesis beyond the logical one.

REFERENCES

[1] D. Appello et. al, "Exploiting programmable BIST for the diagnosis of embedded memory cores", IEEE International Test Conference, 2003, pp. 379-385.

[2] S. Boutobza et. al, "Programmable memory BIST", IEEE International Test Conference, 2005, pp. 1155-1164.

[3] R. Treuer, and V.K. Agrawal, "Built-In Self Diagnosis for Repairable Embedded RAMs", IEEE Design and Test of Computers, Vol. 10, No. 2, June 1993, pp. 24-33.

[4] T. Bergfeld, D. Niggemeyer, E. Rudnick, "Diagnostic testing of embedded memories using BIST", IEEE Conference on Design, Automation and Test in Europe 2000, pp. 305 –309.

[5] D. Appello et. al, "Cumulative Embedded Memory Failure Bitmap Display and Analysis", 13th IEEE Symposium on Design and Diagnostics of Electronic Circuits and Systems 2010,pp. 255-260.

[6] A.J. van de Goor, "Testing Semiconductor memories: Theory and Practice", ComTex Publishing, Gouda, The Netherlands, 1998