Cumulative Embedded Memory Failure Bitmap Display & Analysis

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

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

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

***Abstract*- An effective silicon debug and diagnosis process has to be supported by on-chip hardware structures, stimulation equipments and software tools for analysis.**

In this paper, the characteristics of a software tool for memory failure analysis are presented; this tool takes into account the memory topology and the executed memory test, and returns both syndrome and shape-based failure statistics. Furthermore, it allows the cumulative analysis over many memory cuts inside a die, a wafer or a lot.

The results obtained for embedded SRAMs tested using March test algorithms are presented, demonstrating the capability of the tool in underlining manufacturing process weaknesses and systematic constructive marginalities.

I. Introduction

An effective silicon debug and diagnosis process has to be supported by on-chip hardware structures, suitable stimulation equipments and software tools for analysis.

By means of activating the capabilities of on-chip Built-In Self-Test (BIST) modules, a memory can be tested by applying several test algorithms; the memory test algorithm selection mainly depends on the defects to be tackled; most silicon-debug- and test- oriented BIST engines are programmed accordingly to this decision, possibly also providing diagnostic information [1].

Test algorithm programming, launching, and monitoring during the test process are managed through the test equipment that is finally getting a set of information related to test execution. At the end of the test such information is downloaded to a host machine and then elaborated to isolate failure and to possibly provide their explanation.

Such an off-line analysis has to be performed by software tools taking into consideration the major needs for memory silicon debug and diagnosis, which are the capability of:

- relating the set of extracted faults to the physical topology of the investigated memory to ease the failure analysis
- associating each failing location to the set of test steps producing an error, and recognizing the shapes of the failing areas, such as rows, columns, pairs, clusters, to work out preliminary fault model hypothesis
- enabling the cumulative analysis of a population of embedded memory cuts in order to identify systematic marginalities due to imperfections in the design, manufacturing process, or even due to component handling during test and packaging phases.

This paper details the features that a software tool has to include to tackle the aforementioned issues. In summary, it illustrates how to parameterize the memory physical characteristics, store a minimal amount of information regarding the fault set produced by running a memory test, and which fault grouping strategy is significant to achieve a meaningful fault investigation for a single or a set of devices.

Examples and results reported in the following paragraphs are related to embedded SRAM memories, and a tool for the analysis of memories integrated into System-on-Chip is described. Results gathered on an industrial case study demonstrate the effectiveness of the tool in a real scenario.

II. Memory Diagnosis Background

A briefly introduced, a memory diagnosis process involves several components. Figure 1 illustrates a complete flow for memory failure retrieval and analysis.

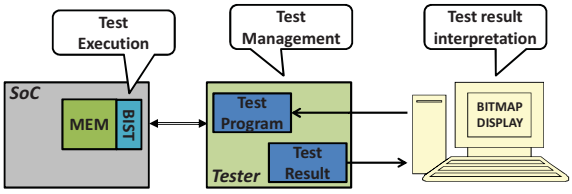

Fig. 1: memory diagnosis flow components

At SoC level, BIST modules enable at-speed memory test application; test selection may be fixed [3] or programmable [4]; BISTs functionalities can take into account diagnosis [5] by storing a complete, or partial, set of observed failure behaviors.

BIST functionalities are controlled by the Tester; it stores a test program that can be constituted by a simple set of commands letting the BIST running or by a more complicated test description including memory test code upload on BIST in case it is programmable and execution parameter manipulation. In this latter case the diagnosis is performed by the application of consecutive steps, eventually determined by the current observed result. The test is therefore also in charge of the temporarily storage of the results of test until it is not completed.

The results collected on the tester are finally retrieved by the host computer to permit their analysis by test engineers and failure analysts. This analysis is crucial to individuation of the failure mechanisms; bitmap tools showing where and how failures appeared are an absolute need for quickly align over suspect memory zone, possibly providing also an initial hypothesis about the fault model.

To be effective towards these issues, bitmap analysis tool should be able to join all the information related to the SoC and Tester characteristics. Figure 2 shows the conceptual view for a memory analysis bitmap display tool.

978-1-4244-6610-8/10/$26.00 ©2010 IEEE

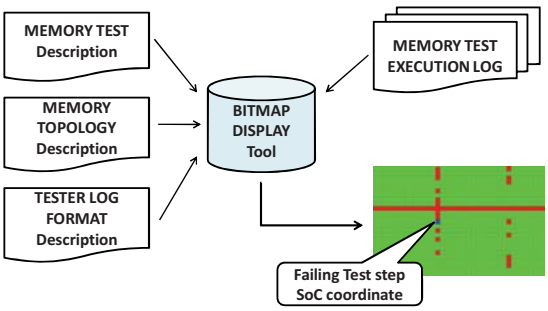

Fig. 2: conceptual view for Bitmap Display tool

III. PROPOSED ANALYSIS FLOW

The proposed methodology for an effective memory failure analysis consists in a set of steps finally allowing displaying the topology of the failing data and providing keys for the following steps of the failure analysis. To achieve this goal, the software environment elaborates the information returned by the tester, basing on a set of parameters describing the physical memory organization and the executed march test.

The implemented data structures permit to minimize the processing effort and the amount of required memory; such a characteristic permits to quickly elaborate fail statistics over a large base of faulty devices.

A. Memory representation

The first outcome of an effective tool for memory analysis 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.

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 1 - 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 3 illustrates this basic BIT ORDER sequence information.

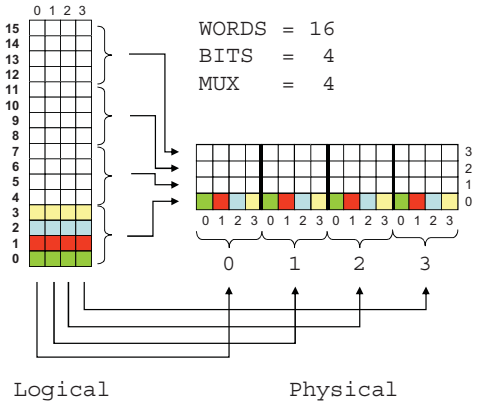

Fig.3: basic mapping between logical and physical SRAM view

Additionally, physical row composition may be completed by a couple of SRAM specific mirror parameters (BLOCK MIRROR and BIT MIRROR). The former specifies if any block sequence order inversion is physically implemented, the other concerns the sequence of bits inside a block. Figure 4 shows an example with BLOCK MIRROR and BIT MIRROR different than 1.

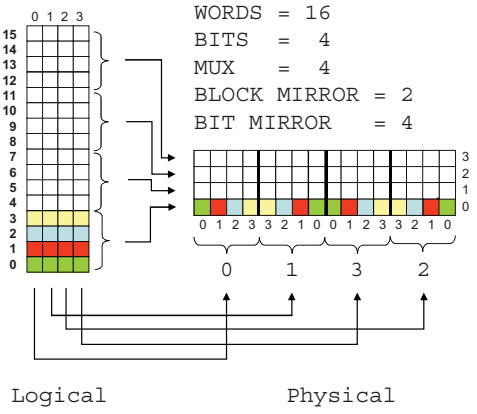

Fig. 4: view with block and bit mirrors

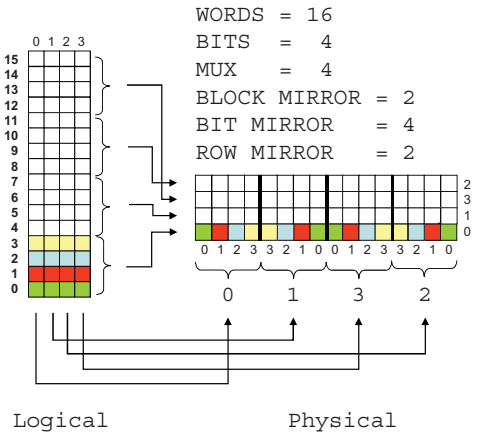

Fig. 5: logical to physical view with all mirrors

Another mirror parameter that can be considered for embedded SRAMs is related to physical row ordering. This parameter may be called ROW MIRROR. An example is shown in figure 5.

To succeed in the conversion from the logical view to the physical one, the software environment has to include a Descrambler module able to translate a logical address into a set of physical locations by taking into consideration the Memory and Scrambling parameters. This capability permits to easily focus the attention on a specific point in the memory map during a failure analysis process.

Based on this goal, another crucial capability that the tool should include is the memory positioning information; this aspect is particularly important in the SoC since embedded memories may be rotated (+90° , -90°) and/or flipped bottom-up. Figure 6 shows some of the most significant positioning configurations starting from the embedded SRAM shown in figure 5.

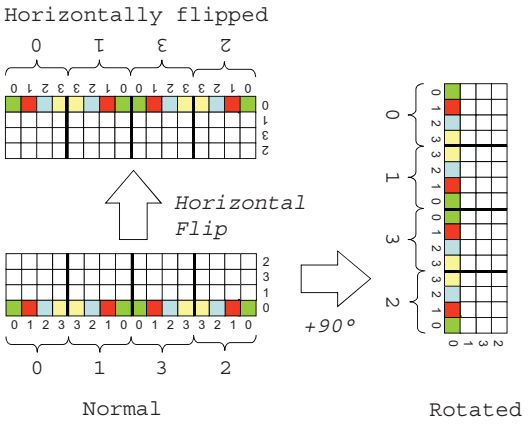

Fig. 6: rotated and flipped SRAM configurations

The conceptual schema of a Descrambler SW module is reported in figure 7. Having such a module enables avoiding the memorization of the full memory such as a matrix.

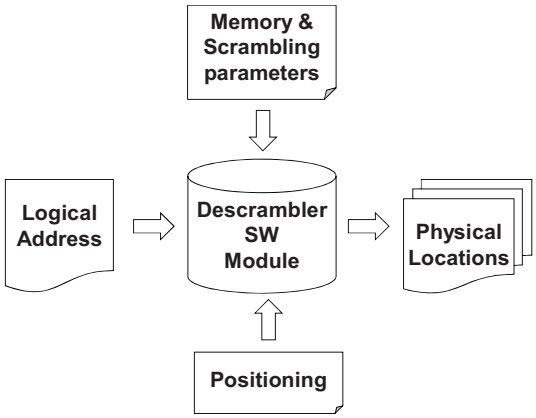

Fig. 7: address translation process

B. Failure Bitmap storage

The Failure Bitmap is the representation of the memory failures on the physical map of the analyzed memory cut. To build such a graphical vision of a faulty component, two major problems have to be considered:

2. the access to the memory is usually performed in a logical manner, therefore returned faulty addresses have to be translated into the physical ones.
3. in many cases, during memory test execution more than one information is returned for a single faulty location.

The former issue can be partly solved by SW modules such as the Descrambling one, but more information may be needed about the applied algorithm. As well, the latter point requires the knowledge of the memory algorithm, thus allowing the compaction of the information retrieved along its execution which relate to a single location.

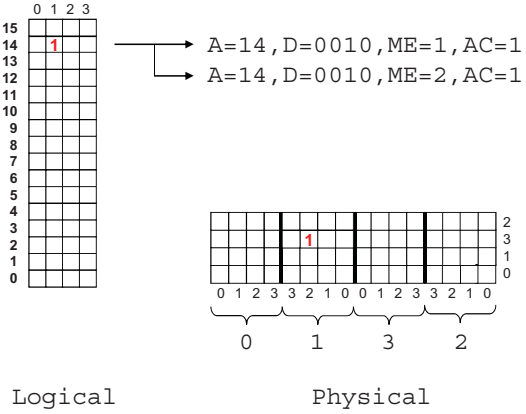

Fig. 8: BIST responses and logical-physical fault location

Example 2 **-** Let consider the embedded memory drawn in figure 5. In case the physical location at Block 1, Bit 2, and Row 3 is stuck to 1 and the memory test executed is the March test [2]:

$$\{\uparrow(w0);\ \updownarrow(R0,W1,R1,W0);\ \downarrow(R0)\}$$

Two failure information items will be returned, one related to the execution of the second March element (ME) at the first memory access (AC), the other related to the third one, again at the first memory access. The extracted logical address is in both cases 13 with read data 0010. This particular faulty case is illustrated in figure 8.

A suitable tool structure solving these problems is illustrated in figure 9; it includes the Descrambler, complemented by an Expected Data Calculator module.

The main result of the work of the two modules is a list of physical locations, each one enriched with a collection of information concerning the failing memory test steps. This data structure is reported in figure 8.

Such a representation of the failing location has a double benefit. It is oriented to the physical memory structure, thus easing the drawing of the failure bitmap; furthermore, it is a compact representation that occupies a smaller space on disk and in RAM during display with respect to the memorization of the complete memory matrix.

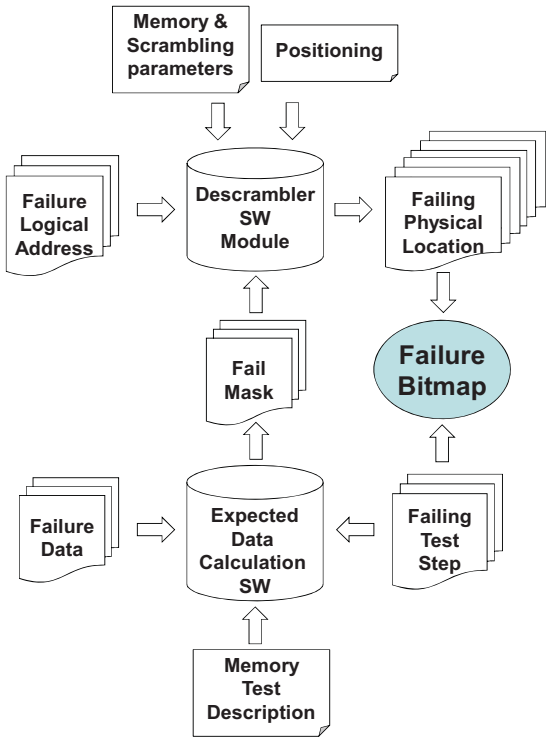

Fig. 9: conceptual working principle of the proposed framework

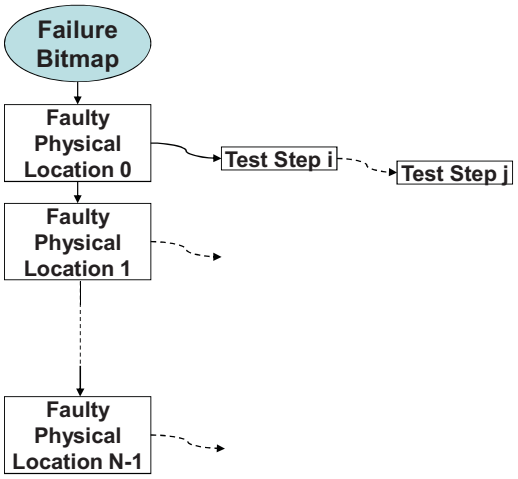

Fig. 10: failure bitmap data structure

Such a structure is also suitable to perform two types of analysis (and their combination).

- **Syndrome grouping analysis:** it consists in considering the failing mechanism of each single cell, and grouping those showing the same set of failing test steps. This analysis consists in reordering the Faulty Physical Location list according to the list of Test Steps; the cost is in the order of $O(N^2)$ if the number of test steps is negligible with respect to the number N of failing locations.
- **Shape-based analysis:** it consists in recognizing a set of predefined shapes in the Failure Bitmap, such as failing rows (partial or complete), columns, pairs, clusters, etc. Most of the classification can be achieved by simply ordering the Faulty Physical Location list according to the physical row/column address and then scanning it. The cost of this operation is of order $O(2N+\log_2(N+1))$.

It is also possible to merge the two types of analysis by partitioning the failure bitmap by shape first, then by syndrome.

Example 3 - Let consider the embedded memory drawn in figure 5 and the March test used in the example 2. A more complex fault scenario is reported in figure 11, including a pair of faulty cells both stacked at 1 and a column defect with all the cells stacked at 0.

In this example, the logical and physical representations are shown; the BIST response is referred to the logical addressing mode, therefore its output has to be converted as shown in the following failure bitmap obtained for the observed faulty configuration.

In this case the failure bitmap is ordered by column, therefore allowing the identification of the failing column. The two bits constituting the pair are not immediately classified, but they will after reordering the failure bitmap by row.

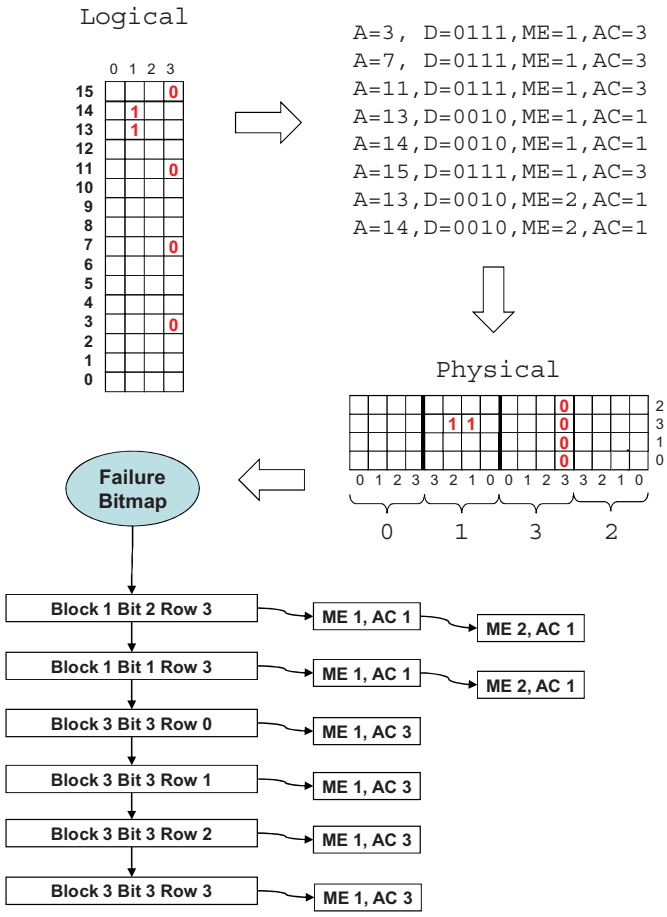

Fig. 11: example with pair and rows

IV. Cumulative Analysis

When tackling effective failure analysis, displaying and grouping failures inside a single memory is not sufficient any more, and the analysis must be extended to a larger set of memory devices.

Useful inspections working out systematic problems in the manufacturing process require observing a population of failing memories. Effective population compositions are:

- **Horizontal population**: it corresponds to accumulating information about the failures of all the memories included in the same wafer; eventually, the inspection can be reduced to accumulate failures in a wafer region.
- **Vertical population**: it corresponds to accumulating the information about failures over many wafers once a wafer coordinate has been selected; eventually, the vertical inspection can be done on a set of wafer coordinates.

Figure 12 illustrates the population composition. While horizontal inspection enables identifying systematic design weaknesses, vertical inspection provides also information about equipment induced failures such as those due to component handling or misalignments during mask and metal deposition process. Cumulative analysis can be done at wafer and singulated component level, assuming in the latter case that traceability information is available to retrieve the wafer and the die position for each single component.

In case of accumulated information, shape-based analysis has to be performed separately on every single failure bitmap.

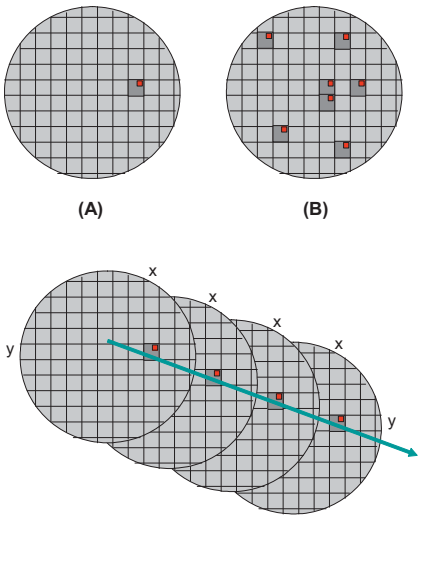

Fig. 12: (A) single (B) horizontal and (C) vertical cumulative analysis

V. Experimental Results

A tool for the front-end analysis of embedded SRAM include in SoCs was implemented in JAVA, following the aforementioned concepts.

Such a tool is able to load information about the memory topology and SoC composition. It considers the output of a programmable BIST engine executing March tests, which functionalities are activated by a tester saving results in the Motorola SRecord format. Figure 13 shows the classification tree built after reading wafer test results; In order to evaluate its performance and capabilities, the tool was used to analyze a population of SoCs belonging to 2 wafers.

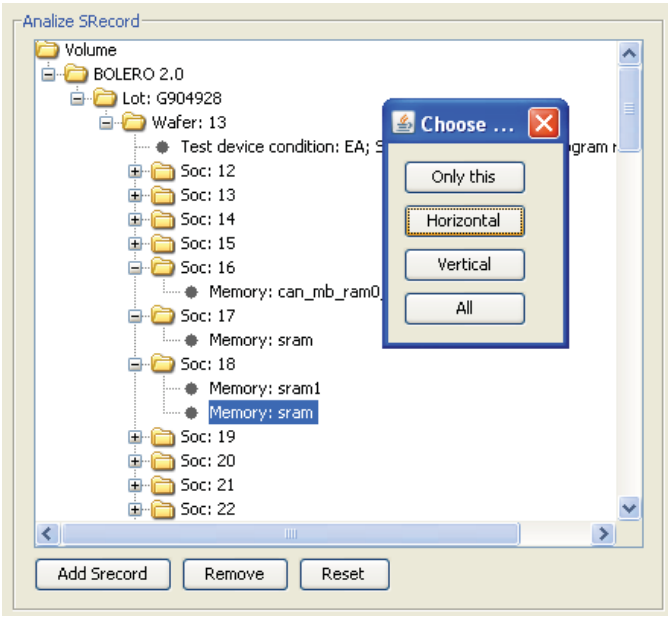

Fig. 13: memory classification tree

In terms of elapsed CPU time, visualizing the cumulative failure bitmap of 58 instances of a 20KB sized embedded SRAM including about 12,000 faults required about 800ms, while statistics calculation required about 1,5s. The amount of main PC memory required to maintain the failure is about 10MB.

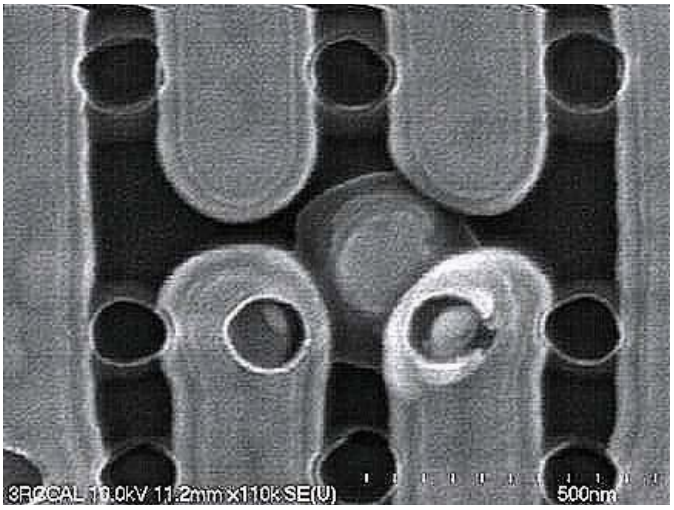

Fig. 15: identified defective mechanism, a stuck-open fault

In terms of the resulting bitmap display is shown in figure 14; the cumulative analysis for the SRAM memory core visualizes here a set of recurrent shapes, including partial and jeopardized failing row. The optical inspection leaded by this graphical hint enabled individuating the recurrent defect corresponding to a short metal connection, which is shown in figure 15.

VI. Conclusions

This paper illustrates the software tool characteristics needed to obtain an effective memory silicon debug and diagnosis process. The described features were implemented in a tool developed in JAVA, suitable for the analysis of embedded SRAMs included in a SoC and tested via a BIST engine at wafer level. The major result of this work is the description of an integrated hardware-software approach for advanced analysis capability.

VII. References

[1] D. Appello, P. Bernardi, A. Fudoli, M. Rebaudengo, M. Sonza Reorda, V. Tancorre, M. Violante, "Exploiting programmable BIST for the diagnosis of embedded memory cores", IEEE International Test Conference, 2003, pp. 379-385

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

[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] S. Boutobza, M. Nicolaidis, K.M. Lamara, A. Costa, "Programmable memory BIST", IEEE International Test Conference, 2005, pp. 1155-1164

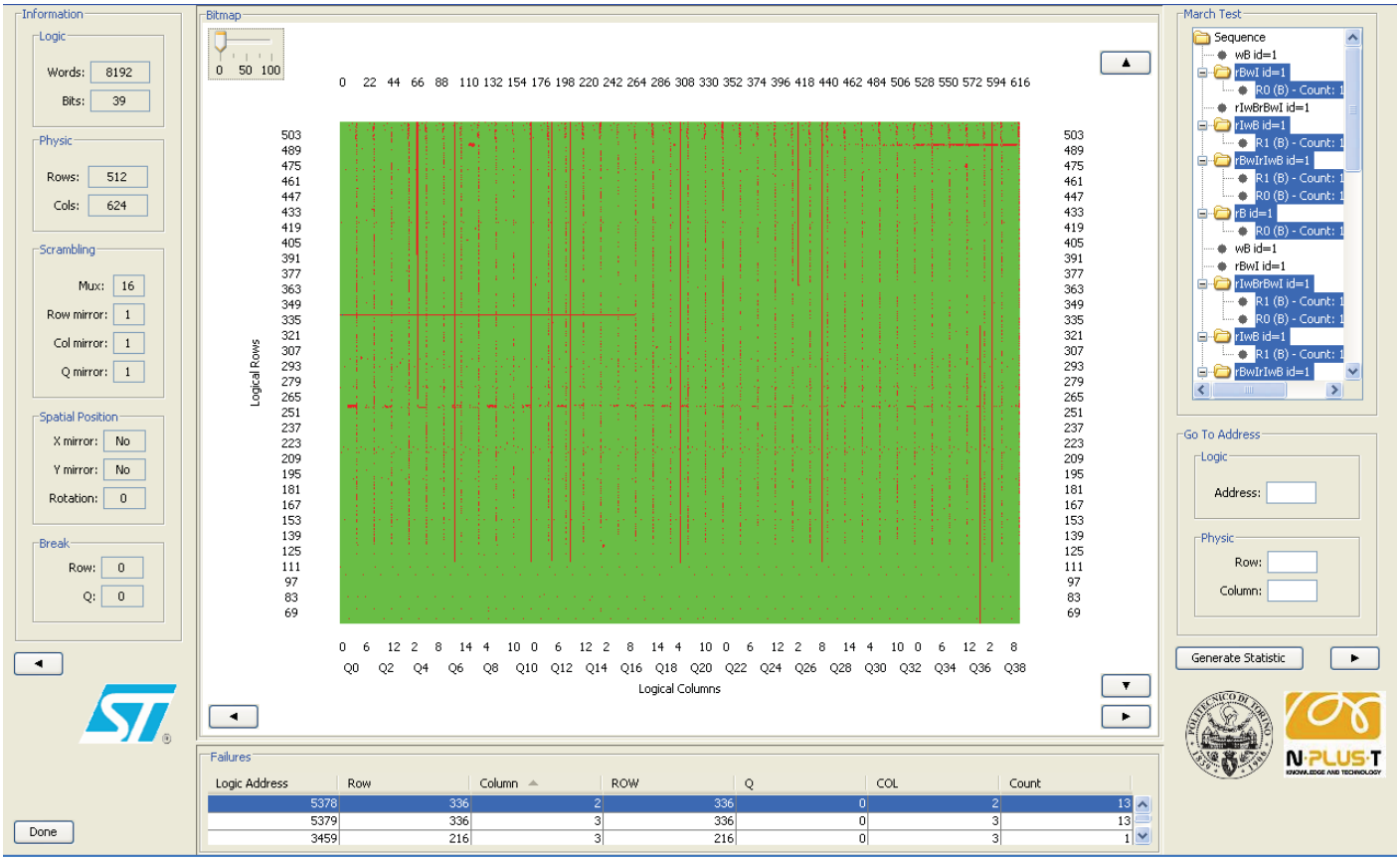

Fig. 14: failure bitmap display window