TIMA Lab. Research Reports

Test Algorithms for ECC-based Memory Repair in Ultimate CMOS and Post-CMOS

Panagiota Papavramidou, Michael NICOLAITIS

ISRN: TIMA-RR--2015/01--FR

TIMA Laboratory
46 Av. Félix Viallet, 38031 Grenoble cedex, France
Test Algorithms for ECC-based Memory Repair in Ultimate CMOS and Post-CMOS

Panagiota Papavramidou, Michael Nicolaidis
TIMA (CNRS, Grenoble INP, UJF)

Abstract. In modern SoCs embedded memories should be protected by ECC against field failures to achieve acceptable reliability. They should also be repaired after fabrication to achieve acceptable fabrication yield. In technologies affected by high defect densities, conventional repair induces very high costs. To reduce it, we can use ECC-based repair, consisting in using the ECC for fixing words comprising a single faulty cell and self-repair to fix all other faulty words. However, as we show in this paper, for high defect densities the diagnosis circuitry required for ECC-based repair may induce very large hardware cost. To fix this issue, we propose and develop a new family of memory test algorithms that exhibits the “single-read double-fault detection” property. This approach gains interest in ultimate CMOS and post-CMOS technologies, where the defect densities are expected to increase significantly, and/or in very-low power design, as very-low voltage sharply increases defect densities.

Keywords: Design for Yield, Design for Reliability, Memory Repair, Memory Test Algorithms, Error Control Codes.

I. INTRODUCTION

Embedded memories occupy the largest part of modern SoCs and include even larger proportions of transistors. As memories are designed very tightly to the technology limits, they are more prone to failures than other circuits. Thus, they concentrate the large majority of fabrication defects, affecting yield adversely. Hence, memory Built-In Self-Repair [1-9] became early mandatory for maintaining acceptable fabrication yield. Also, it was early predicted [10] that achieving acceptable reliability levels is one of the most critical issues for late CMOS. Indeed, field failures (soft-errors caused by neutrons and alpha particles, weak-cell faults activated during very-low voltage modes, circuit aging and wearout) are critical concerns in memories. Thus, during the last decade, ECC became mandatory for maintaining acceptable reliability levels [11-16]. As ECC can cope with both field and fabrication faults, we can use it for mitigating both fault types and reduce cost. In particular, ECC can be used to cope with fabrication faults affecting a single cell of a memory word. For low fabrication-fault rates, ECC could completely eliminate the repair circuitry. For high fabrication-fault rates we can use word repair to fix only the words containing more than one faulty cells, while words containing one faulty cell can be fixed by ECC. Disposing techniques able to repair memories for high fault rates is increasingly suitable as ultimate-CMOS and post-CMOS nanotechnologies are expected to be affected by high defect densities due to the steadily worsening process, voltage, and temperature variations, and aging-induced circuit degradation. In addition, aggressively reducing voltage levels is highly desirable, since reducing power dissipation as much as possible is one of the most stringent requirements in future process generations. But such reduction will have severe impact on fault rates. Thus, being able to repair memories for as high fault rates as possible is highly desirable for future technologies, as it will allow both, cope with high defect densities intrinsic to these technologies and reduce aggressively power.

Combining ECC with word repair is highly efficient for repairing memories affected by high defect densities:
- As shown in [17], combining word repair and ECC is the only cost-effective solution (in terms of area cost) for achieving acceptable fabrication yield for high defect rates.
- As stated in [18] and confirmed in [19] “the number of repairable faults dramatically increases by combining the ECC and redundancy techniques together”.

However, as we show in the present paper, previous work on ECC-based memory repair has neglected one important issue: the hardware cost reduction gained by using ECC-based repair can be lost due to diagnosis requirements. To cope with this problem, in this paper we propose and develop a new family of memory test algorithms that exhibit a property that we termed as “single-read double-fault detection”.

II. DfX REUSE STRATEGY

During the last decade, soft-errors caused by atmospheric neutrons and alpha particles became a major reliability concern in modern electronic systems [11][12][20]. This trend has led to the systematic protection of memories by means of ECC, most often implemented as single-error correction - double-error detection (SEC-DED) codes. As process scaling has increased the susceptibility to multi-cell upsets MCUs), that outperform the capabilities of SEC-DED codes, interleaving is also used to guaranty that at most one cell upset can affect the same memory word (single-bit-upset – SBU). The typical soft-error rate (SER) per Mbit SRAM at 28nm planar bulk CMOS is about 180 FIT for single-cell upsets (SCUs) and 10 FIT for MCUs. The introduction of finFET is accompanied by sharp SER reduction (SER in 14nm finFET process is about 12 FIT for SCUs, and 0.6 FIT for MCUs), but memory soft-error protection remains necessary as increasing chip complexities maintain significant SER at chip level. Process scaling is accompanied by sharp critical charge reduction, increasing soft error sensitivity, but this trend is counterbalanced by the reduction of sensitive volume. Thus, the SER per Mbit SRAM in future finFET nodes is expected to be constant or slightly reduced. However, the SER per chip will be increased significantly, making memory soft-error mitigation increasingly mandatory.

As DfX techniques are proliferating (Design for Test, Design for Debug, Design for Yield, Design for Low-Power,
Design for Reliability …), the amount of hardware dedicated to non-computational purposes grows substantially. It is therefore mandatory to combine different DFX techniques to moderate their impact on area, power and/or performance. As ECC is already implemented in memories for mitigating soft-errors, it may be reused to reduce the cost of ECC-based repair. The critical issue for this kind of reuse is however the potential reduction of soft-error mitigation efficiency. As words containing one faulty cell are not repaired in ECC based repair, upsets affecting such words may invalidate SEC-DED protection. As usually, we consider that the memory is implemented by using adequate scrambling, so that, MBUs (multi-cell upsets) will impact at most one cell in any memory word. Thus, upsets could only induce double errors in a memory word containing one faulty cell. As these errors are detected by the SEC-DED codes, reliability is not affected. On the other hand, these errors can not be corrected by SEC-DED codes, but they can be fixed by check-point based rollback recovery, employed in most cross-layer reliability approaches under development for addressing the reliability of upcoming process generations. Then, the only critical issue concerns the frequency of error recovery activations. However, as we show next, this frequency is very low and will induce negligible impact on system availability. In this work we consider high fault rates ranging from $10^{-5}$ to $10^{-3}$ probability for a memory cell to be faulty. In the less favourable case ($10^{-2}$ faulty-cell probability), corresponding to 2 or 3 orders of magnitude higher faulty-cell rates than in current technologies, considering memory words of 39 bits (32 data bits and 7 SEC-DED check bits), gives a probability for a memory word to contain one faulty cell equal to $0.999^{39} \times 10^{-3} \times 39$. In a very large SRAM of 100 Gbit capacity, this gives an average of $0.999^{38} \times 10^{-3} \times 39 \times 10^{11} / 39$ words containing a single faulty cell. As the SER per Mbit is slightly reduced with process scaling but MCU rate increases, considering 8 FIT for SCUs and 2 FIT for MCUs per Mbit SRAM after several finFET process generations is a reasonable projection. Considering a mean number of 6 cells hit by an MCU and also that interleaving is employed as usually to ensure that an MCU affects only a single cell of the same memory word, we find that the per year rate of single-cell upsets affecting a memory word containing a single faulty cell is equal to $(0.999^{38} \times 10^{-3} \times 39 \times 10^{11} / 39) (8+2x6) (38/10^6) (10^3 \times 365.25 \times 24) = 0.641$, where: the first parenthesis gives the average number of words containing one faulty cell in the 100 Gbit SRAM; the second parenthesis gives the mean number of cells in a 1 Mbit SRAM affected by upsets in 10^6 hours (1 FIT = 1 event in 10^6 hours), considering 8 FIT per Mbit for SCUs, 2 FIT per Mbit for MCUs, and a mean of 6 cells affected by an MCU; the third parenthesis gives the number of fault-free cells of a word containing one faulty cell (as only hits affecting a fault-free cell of the word can produce double errors), divided by the number of cells of a 1 Mbit memory; and the fourth parenthesis gives the number of events per year per FIT. To produce a double error, the upset should occur when the correct value of the faulty cell corresponds to its sensitized value. The probability for this to happen is 0.5. Furthermore, for most faults some other conditions should also be activated for sensitizing the faulty cell. Hence, the faulty-cell sensitization probability is lower than 0.5. The occurrence probability of the double error is further reduced, due to functional derating issues: if the affected word is not used by the application, or if it is written before be read, no double-error will be observed. Thus, the probability that a faulty memory word affected by an upset produces a double error during a read operation is quite lower than 0.5. In the following, we will use the value 0.5 which is quite pessimistic.

Then, for a very large memory (100 Gbit capacity), less than 0.32 events per year can produce a double error, which is not corrected but it is detected by the ECC. Therefore, in the worst-case scenario of a $10^{-3}$ faulty-cell probability, the reuse of the ECC for ECC-based repair in a large SRAM of 100 Gbit capacity will induce less than one error-recovery interruptions at every 3 years, which is insignificant. If we consider a still high but less excessive faulty-cell probability equal to $10^{-4}$, reusing ECC for ECC-based repair in a 100 Gbit SRAM will induce less than one error-recovery interruption at every 30 years. In a future single-chip massively-parallel tera-device processor consisting in 4000 processing nodes using 250 Mbit memory per node (a total of 1 Terabit memory), and employing ECC-based repair as envisioned in the CELLS framework [21-23], for the $10^{-4}$ faulty-cell probability each node will experience less than 1 interruptions for error recovery at every 1250 years, while for the $10^{-3}$ faulty-cell probability each node will experience less than 1 interruptions for error recovery at every 12000 years. Furthermore, as the CELLS framework performs check-point-free error recovery by means of an innovative approach exploiting hierarchical task allocation in the multiprocessor grid [24], performance lost induced by check-pointing is also eliminated. Thus, ECC-based repair reusing ECC implemented for soft-error mitigation is a winning strategy in all above scenarios.

Due to the aggressive process scaling, ultimate CMOS and post-CMOS technologies are expected to be affected by high densities of manufacturing and aging-induced faults, as well as high power and temperature densities. Thus, they will require powerful techniques for off-line repair, runtime fault tolerance, and low power. Our developments of efficient memory repair for high fault-densities are motivated by the goal to reuse the hardware resources of ECC-based repair in order to address all the above critical issues. First, the expectation of steadily worsening process variability and aging-induced circuit degradation in upcoming process nodes makes mandatory the development of efficient memory self-repair for high fault-densities, in order to improve manufacturing yield after fabrication and subsequently extend the life of the circuit by repairing aging-induced faults. Furthermore, disposing a self-repair technique for fault densities much higher than those required for manufacturing and aging-induced faults, will also allow drastically reducing power dissipation. Indeed, in this case we can reduce aggressively the operating voltage (Vdd) in order to achieve drastic reduction of power dissipation (i.e proportional to the square of the reduction of Vdd), and repair the memory cells that exhibit faulty behavior due to the reduced Vdd. Ultimately, reliability issues can also be caused by aging-induced faults that occur during the execution of an application. Such faults may affect the correct execution of the
application during the time that elapses between their instant of occurrence and the next test and repair session. Disposing a self-repair technique for fault densities even higher than those required for the three classes of faults considered above, will allow solving this reliability issue by testing the memory under more stringent conditions (lower voltage and/or higher speed) than the worst conditions used during the execution of any application. This is because such tests can detect and proactively repair those memory cells that do not yet exhibit faulty behavior, but are already weakened and have increased chances to become faulty before the next test and repair session.

III. AREA, POWER, AND DIAGNOSIS ISSUES

A. Area and Power Issues

RAMs are repaired by replacing faulty regular units by fault-
free spare units. The resulting cost corresponds to the cost of the spare units and of the circuitry used to control the unit replacement. For low defect densities, the principal cost source are the spare units. However, in some repair schemes, as defect densities increase the cost of the circuitry controlling the replacement can become more important. Indeed, there are two schemes for replacing regular units by spare units.

The first stores into dedicated registers the positions of the faulty units gathered during the test phase. Then, dedicated circuitry decodes this information to generate the control signals of a set of MUXes used to disconnect faulty regular units and connect in their place fault-free spare units. However, for high defect densities, we need to use a large number of small spare units to repair an even larger number of small regular units. As both the number R of regular units and the number S of spare units increase, the complexity of the hardware implementing the MUXes increases exponentially, since we need R MUXes of S+1 inputs and 1 output each. The circuitry controlling the MUXes also increases exponentially as it has to generate Rx(S+1) signals. In addition, the hardware cost increases more rapidly than the exponential increase of the number Rx(S+1) of the generated signals, as the complexity of the combinational function generating each of these signals is not constant but increases as Rx(S+1) increases. Thus, the exponential increase of the hardware cost and power dissipation is of higher order than Rx(S+1), and MUX-based repair is not scalable to high defect densities.

The second scheme uses a CAM for storing the addresses and data of faulty units. This approach is scalable in size for increasing defect densities, as the CAM size is proportional to the number of faulty units, resulting in linear area increase. Thus, CAM based repair is preferable for high defect densities. However, another critical issue is power, as CAMs are power hungry. Using ECC-based repair reduces drastically the size of the CAM (as we only repair words containing more than one faulty cells), resulting in drastic reduction of area and even more drastic reduction of power penalties.

B. Diagnosis Issues

For implementing ECC-based memory repair it is necessary to identify after fabrication the words that comprise more than one faulty cells. It is also important to identify such words in the field, in order to cope with faults occurring during circuit life due to aging-induced circuit degradation. However, if a word contains more than one faulty cells, then, memory test algorithms do not guaranty that all of them are detected within the same read operation. Indeed, one fault can be detected by a read belonging to a march element and another fault can be detected by a read belonging to another march element. Thus, with existing memory test algorithms, each time a read detects a faulty cell we need to store the address of the faulty word as well as the position(s) of the faulty cell(s), and then, use an algorithm for determining the words containing several faulty cells detected at different instants of the test algorithm. The most efficient solution for storing this information and identifying the words containing multiple faulty cells is to use a CAM in which we store the faulty addresses in the tag-fields and the positions of the faulty bits in the associated data fields. As at the instant we detect one faulty cell in a memory word we don’t know if this word contains also other faulty cells that will be detected later, then, each time we detect a faulty word we are obliged to store it in the CAM, whether it contains one or more faulty cells. Thus, we need a CAM of the same size as in the case where we do not exploit ECC for repair purposes. In high defect densities, the area and power cost of this CAM will be very high. Therefore the benefits expected by the use of ECC-based repair are lost due to diagnosis requirements. Of course, it is possible to eliminate the diagnosis CAM if we shift out of the chip the faulty addresses and faulty cell positions and use the intelligence of an external tester for performing the diagnosis. However, current and future chip complexities make this approach inefficient. Furthermore, using an external tester will prevent test and repair for field failures, which is necessary as the rate of aging-induced faults is becoming high in advanced process nodes.

To cope with these issues we developed an approach, which eliminates altogether the need of diagnosis hardware by introducing a new kind of test algorithms. These algorithms have the property that if a memory word contains two or more faulty cells, there is a read operation in the algorithm that detects at least two of them: single-read double-fault detection (SRDF) property. Thus, we store a memory word in the CAM only when a single read detects two or more faulty cells in this word. Hence, no memory word containing only one faulty cell is stored. As a consequence, there is no CAM size increase due to diagnosis issues. Algorithms exhibiting this property will be referred as SRDF test algorithms.

The rest of the paper addresses the highly challenging task related with the development of SRDF test algorithms, which due to the SRDF constraint, are substantially more difficult to develop with respect to any existing test algorithm. To address this challenge we developed a formal framework described in the following sections.
IV. MEMORY TEST ALGORITHMS FOR ECC-BASED REPAIR

As highlighted in the previous sections, the benefits gained by using ECC-based repair can be lost due to diagnosis issues. Here we address the development of SRDF test algorithms enabling eliminating this issue. These algorithms guarantee the single-read double fault detection property. That is: if a memory word contains two or more faults, there is a read operation in the algorithm that detects at least two of them. We first treat the case of words affected by two faults (propositions 1 through to 8), and then the case of words affected by three or more faults (propositions 9). We start with an illustration of the approach for state faults [26]. Then, we propose algorithms for more complex faults.

Let us consider a simple march test algorithm detecting state faults: \( \{ \mathcal{I}(W0) ; \mathcal{I}(R0) ; \mathcal{I}(W1) ; \mathcal{I}(R1) \} \). Here W0 means that a write is performed using the all 0’s vector as data, and R0 means that a read is performed with expected value the all 0’s vector. In this algorithm a state fault SF < 0/1/> affecting a cell of any memory word is detected in the 2<sup>nd</sup> march element, while SF < 1/0/> affecting a cell of any memory word is detected in the 4<sup>th</sup> march element. If two SF < 0/1/> faults affect the same memory word both are detected by a single read (i.e. when this word is read in the 2<sup>nd</sup> march element). Similarly, if two SF < 1/0/> faults affect the same memory word both are detected in a single read (i.e. when the word is read in the 4<sup>th</sup> march element). However, if a SF < 0/1/> and a SF < 1/0/> affect the same word, then, the first fault is detected in the 2<sup>nd</sup> march test sequence and the second fault is detected in the 4<sup>th</sup> march test sequence. But we need to detect in the same read any two state faults affecting any two cells of the same memory word. The solution is to use the algorithm \( \{ \mathcal{I}(Wv) ; \mathcal{I}(Rv) ; \mathcal{I}(W \widehat{V}) ; \mathcal{I}(R \widehat{V}) \} \), obtained by replacing in the algorithm described above the all 0’s vector by a vector Vi and the all 1’s vector by \( \widehat{V} \). Then, we execute this algorithm k times using each time a different vector \( V_i \), \( i \in \{0, 1, \ldots, k\} \). This set of binary vectors is selected such that, for any two bit-positions, there is a vector in the set supplying 00 in these positions and another vector in the set supplying 01 in these positions. Let us now consider a memory word comprising two faulty cells. In the above-defined set there is a vector \( V_{00} \) such that the bit positions corresponding to the faulty cells take the values 00. Let us execute the test algorithm using \( V_{00} \) as test data. Then, if the two faults are SF<0/1/> both of them are detected when the faulty word is read in the 2<sup>nd</sup> march element of the algorithm; and if the two faults are SF<1/0/> both are detected when the faulty word is read in the 4<sup>th</sup> march element of the test algorithm. Furthermore, in the above-defined set there is a vector \( V_{01} \) such that the bit positions corresponding to the faulty cells take the values 01. Let us execute the test algorithm using \( V_{01} \) as test data. Then, if the one fault is SF<0/1/> and the other is SF<1/0/> both are detected when the faulty word is read in the 2<sup>nd</sup> march element; and if the one fault is SF<1/0/> and the other is SF<0/1/> both are detected when the faulty word is read in the 4<sup>th</sup> march element. Thus, executing the algorithm for the above-defined set of vectors \( V_i \) guaranties the single-read double-fault detection property for all double state faults affecting the same word. Such a set consists in \( k=\lfloor \log_2 m \rfloor +1 \) vectors \( V_i \) (m being the size of the memory word) [27], and its creation is simple:

- We generate the set of all 2<sup>k</sup> binary numbers of k bits. This set contains \( 2^{k-1} \) pairs of complementary binary numbers.
- We eliminate any one of the two numbers of each pair, to obtain \( 2^{k-1} \) binary numbers.
- We eliminate \( 2^{k-1} - m \) of these numbers to obtain m binary numbers of k=\( \lfloor \log_2 m \rfloor +1 \) bits.
- We create a kmxk matrix having each of these numbers as a column. The k rows of this matrix give the set of vectors Vi we are looking for.

As an example, for m = 8 the number of vectors \( V_i \) is k = 4. The rows of the 4x8 matrix given below and created as described above, provide a set of 4 vectors \( V_i \) having the above described property.

\[
\begin{array}{cccccccc}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\
0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\
0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\
\end{array}
\]

More complex faults are considered next.

A. Functional Fault Models and Test Conditions

The aim of this section is twofold: present a methodology enabling achieving the single-read double-fault detection property for faults more complex than state faults in order to pave the way for further developments in this domain; propose test algorithms achieving this property for a comprehensive set of faults. Unlinked faults (also known as simple faults) are considered, since the treatment of linked faults is still an open question in the memory test literature. Indeed, as noted in [25] “the fault space for linked faults as well as the required tests remain still to be worked out”. First, we consider single-cell unlinked faults. As concerning multi-cell faults, similarly to [25] in this paper we restrict our analysis to two-cell faults, because they are considered to be the most important class of multi-cell faults. Furthermore, we restrict our analysis to static faults. However, the approach developed hereafter can be used to extend the analysis to dynamic faults too. A systematic classification of all static unlinked functional fault models (FFMs) involving one memory cell (single-cell FFMs) and two memory cells (two-cell FFMs) is presented in [26]. These FFMs are reported in tables 1 and 2. For compactness purposes, in these tables we replaced: the value of the aggressor cell (whether it is 0 or 1) by a; the value of the victim cell (whether it is 0 or 1) by v; the symbol of the transition of the victim cell (whether it is ↑ or ↓) by ◊.

<table>
<thead>
<tr>
<th>#</th>
<th>FFM</th>
<th>Fault Primitives</th>
<th>#</th>
<th>FFM</th>
<th>Fault Primitives</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SF</td>
<td>&lt;v/◊/&gt;</td>
<td>4</td>
<td>RDF</td>
<td>&lt;rv/◊/&gt;</td>
</tr>
<tr>
<td>2</td>
<td>TF</td>
<td>&lt;v wy/◊/&gt;</td>
<td>5</td>
<td>DRDF</td>
<td>&lt;rv wy/◊/&gt;</td>
</tr>
<tr>
<td>3</td>
<td>WDF</td>
<td>&lt;v wyv/◊/&gt;</td>
<td>6</td>
<td>IRF</td>
<td>rv wy/◊/&gt;</td>
</tr>
</tbody>
</table>
Table 2. List of two-cell FFMs

<table>
<thead>
<tr>
<th>#</th>
<th>FFM</th>
<th>Fault Primitive</th>
<th>#</th>
<th>FFM</th>
<th>Fault Primitive</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>CFst</td>
<td>(&lt;a; v, \bar{v} / . &gt;)</td>
<td>3</td>
<td>CFtr</td>
<td>(&lt;a; w, \bar{v} / w &gt;&gt;)</td>
</tr>
<tr>
<td>2</td>
<td>CFds:</td>
<td>(&lt;ta; v, \bar{v} / . &gt;)</td>
<td>4</td>
<td>CFwd</td>
<td>(&lt;a; w, v w / v &gt;&gt;)</td>
</tr>
<tr>
<td>2.1</td>
<td>CFds(a)</td>
<td>(&lt;aw, \bar{v} / . &gt;)</td>
<td>5</td>
<td>CFr</td>
<td>(&lt;a; r, v / v &gt;)</td>
</tr>
<tr>
<td>2.2</td>
<td>CFds(a, b)</td>
<td>(&lt;awa; v / . &gt;)</td>
<td>6</td>
<td>CFdr</td>
<td>(&lt;a; r, v / v &gt;)</td>
</tr>
<tr>
<td>2.3</td>
<td>CFds(a, b)</td>
<td>(&lt;awa; v / . &gt;)</td>
<td>7</td>
<td>CFir</td>
<td>(&lt;a; r, v / v &gt;)</td>
</tr>
</tbody>
</table>

Memory test algorithms covering various combinations of the above faults are presented in [25]. The detection of all of them is achieved at 22n complexity (March SS algorithm [25]). Optimal tests for these FFMs are presented in [28], requiring 19n complexity. In the following, we address test algorithms satisfying the single-read double-fault detection (SRDF) property for these FFMs. That is: if any combination of the above faults affects two or more cells of the same memory word, then, there is always a read operation in the test algorithm that detects at least two of them (single-read double-fault detection property).

The theoretical challenge in developing test algorithms satisfying the SRDF property is far more complex than for conventional tests. Thanks to the unlinked property of the fault models, some simplification of this task can be done by means of a lemma described next. Let us consider a memory word W in which k distinct cells (to be referred as victim cell) are affected by faults, and let k ≥ 2. A victim cell can be affected by multiple FFMs. Thus, a number of FFMs larger than k can affect the k victim cells of W. Let Wf be the set of FFMs affecting the word W.

**Lemma 1:** A test algorithm that realizes each of the conditions i through x given below, guaranties detecting each of the single-cell faults (SF, TF, WDF, RDF, DRDF, IRF), and the two-cell faults of types CFst and CFds, by a some of its read operations performed over the victim cell (referred hereafter as victim-detection read or more simply as detection read):

i. The victim cell is at state v during the victim-detection read (SF faults related condition).

ii. The operations wvw (for WDF faults) or rv (for DRDF faults) is performed over the victim cell before the victim-detection read, and no write is performed over the victim cell between these operations.

iii. The victim-detection read is performed when the victim is at state v (RDF and IRF faults).

iv. The operations vvw is performed over the victim cell before the victim-detection read and no write is performed over the victim cell between these operations (TF faults).

v. There is an instant of the test algorithm preceding the instant of the victim-detection read such that: the aggressor cell is at state a; the victim cell is at state v; and no write is performed over the victim between these two instants (CFst fault).

vi. The operation ta is performed over the aggressor cell at some instant preceding the victim-detection read during which the victim cell has a particular value v, and no write is performed over the victim cell between these instants (CFds(a) faults).

vii. Same as vii with ra replaced by aww (CFds(a, b) faults).

viii. Same as vii with ra replaced by awv (CFds(a, b) faults).

ix. Conditions v, vi, vii, and viii are realized for both a = 0 and a = 1.

x. The test is executed several times using test data that supply to the victim cell both values v = 0 and v = 1 during the realization of each of the above conditions.

Lemma 1 can be proven trivially.

As mentioned earlier, the analysis and derivation of test algorithms achieving the single-read double-fault detection property is far more complex than for conventional tests. Thanks to the unlinked property of the fault models, some simplification of this task can be done by means of a lemma described next. Let us consider a memory word W in which k distinct cells (to be referred as victim cell) are affected by faults, and let k ≥ 2. A victim cell can be affected by multiple FFMs. Thus, a number of FFMs larger than k can affect the k victim cells of W. Let Wf be the set of FFMs affecting the word W.

**Lemma 2:** If a test algorithm TA detects in the same read the faults of any subset Wf of the set of faults W affecting a memory word W, such that the faults of Wf affect at least two victim cells of W, then, TA achieves the single-read double-fault detection property for the faulty word W affected by the faults of the set Wf.

**Proof:** Since the faults are unlinked, the detection of a fault is not masked by the presence of other faults. Thus, if TA detects in a single read the faults of Wf, it will also detect these faults in the presence of all faults of W. As there are at least two victim cells in Wf, this read will detect at least two errors, identifying W as a word affected by at least two faults. QED

Thanks to this lemma, we only need to treat double faults \([f1, f2]\), such that f1 and f2 affect two distinct cells of the same memory word, and each of them is an FFM of the table 1 or 2. Thus, all our proofs will demonstrate the single-read double-fault detection property for double faults only.

**B. SRDF March Test Algorithms**

Based on lemma 1 we propose the March SRDF algorithm (shown in figure 1) that satisfies the single-read double-fault detection property for all single-cell FFMs (SF, TF, WDF, RDF, DRDF, IRF), and two important classes of two-cell FFMs (CFst and CFds). March SRDF1 comprises...
three march elements (M₀, M₁, M₂). M₀, has one operation. M₁ has six operations (M₁₁ through to M₁₆). M₂ has seven operations (M₂₁ through to M₂₇).

**Table 3. Conditions for detecting FFMs in M₂₂**

<table>
<thead>
<tr>
<th>FFMs</th>
<th>Relation @a@v</th>
<th>Sensitization</th>
</tr>
</thead>
<tbody>
<tr>
<td>SF &lt; v/ &gt;</td>
<td>-</td>
<td>M₂₂</td>
</tr>
<tr>
<td>TF &lt; v w/v/ &gt;</td>
<td>-</td>
<td>M₁₅</td>
</tr>
<tr>
<td>WDF &lt; v w/v/ &gt;</td>
<td>-</td>
<td>M₁₆</td>
</tr>
<tr>
<td>RDF &lt; v/ v/ &gt;</td>
<td>-</td>
<td>M₂₁</td>
</tr>
<tr>
<td>DRDF &lt; v/ v/ &gt;</td>
<td>-</td>
<td>M₂₁</td>
</tr>
<tr>
<td>IRF r/v/ &gt;</td>
<td>-</td>
<td>M₂₂</td>
</tr>
<tr>
<td>CFst &lt; a; v/ &gt;</td>
<td>@a@v</td>
<td>M₁₄: a=0 &amp; a=1 ∀V₁</td>
</tr>
<tr>
<td>CFst &lt; a; v/ &gt;</td>
<td>@a@v</td>
<td>M₂: a=0 &amp; a=1 ∀V₁</td>
</tr>
<tr>
<td>CFds &lt; ra; v/ &gt;</td>
<td>@a@v</td>
<td>M₁₃: r0 &amp; r1 ∀V₁</td>
</tr>
<tr>
<td>CFds &lt; ra; v/ &gt;</td>
<td>@a@v</td>
<td>M₁₃: r1 &amp; ∀V₁</td>
</tr>
<tr>
<td>CFds &lt; aw a; v/ &gt;</td>
<td>@a@v</td>
<td>M₁₃: w0 &amp; w1 ∀V₁</td>
</tr>
<tr>
<td>CFds &lt; aw a; v/ &gt;</td>
<td>@a@v</td>
<td>M₁₃: w1 &amp; w1 ∀V₁</td>
</tr>
<tr>
<td>CFds &lt; aw a; v/ &gt;</td>
<td>@a@v</td>
<td>M₁₃: w0 &amp; w0 &amp; w0 &amp; w1 ∀V₁</td>
</tr>
</tbody>
</table>

For CFst faults: march element M₁ writes on the aggressor cell both the 0 and the 1 values regardless to the value of vector V₁. Furthermore, for @a@v no write operation is performed over the victim cell between these writes and the M₂₂ read of the victim cell. Thus, condition v of lemma 1 is realized in M₁ for @a@v and for both values a=0 and a=1 of the aggressor cell. For @a@v, the same holds true for the writes performed on the aggressor cell in M₂. Thus, condition v of lemma 1 is realized for both values a=0 and a=1 also for @a@v. The third column of the table reports these facts for CFst faults. For all other faults, the operations realizing the conditions i, ii, iii, iv, vi, vii, and viii for @a@v and for @a@v are also reported in the third column. As reported in this column, these operations realize the conditions i, ii, iii, iv, vi, vii, viii for both a =0 and a =1, regardless to the value of vector V₁. Thus condition ix of lemma 1 is also realized. Hence, according to lemma 1, executing SRDF1 for a vector V₁ will detect in the read operation M₂₂ of a victim word any fault of the type SF, TF, WDF, RDF, DRDF, IRF, CFst or CFds for the value v supplied by V₁ to the victim cell of this fault. As the set of vectors V₁ is a two-covering set, each of the values 00, 01, 10 and 11 is applied to any pair of bit positions by some vector V₁. Thus, if any two cells of a memory word are affected by any pair of faults of the types SF, TF, WDF, RDF, DRDF, IRF, CFst, or CFds, there is a vector V₁ that enables detecting both faults in operation M₂₂, provided that there are no bit positions coincidences. Furthermore, bit-positions coincidences are not an issue because: on the one hand, as justified earlier this issue does not concern the coincidence of the positions of the victim cells, and on the other hand, all sensitizing operations/states of the aggressor cells are supplied by the test algorithm regardless to the value of vector V₁. Thus, supplying the same value by vector V₁ to coinciding aggressor cells or to coinciding aggressor and victim cells is not an issue. QED

Note that, a 2-covering set of vectors V₁ can be obtained easily by taking the set of vectors used in section II for
satisfying the single-read double-fault detection property for state faults, and adding their complements. For instance for \( m = 8 \) (8-bit words) a 2-covering set is given in the matrix below (the rows of this matrix are the vectors of the 2-covering set, and are obtained by adding the complements of the 4 rows of the matrix generated in section II.):

\[
\begin{array}{cccccccc}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\
0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\
0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\
1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\
\end{array}
\]

The cardinality of the covering set of vectors \( V_i \) generated by the above method is \( k = 2\lceil \log_2 m \rceil + 2 \) (with is the minimal cardinality for a covering set). For 22-bits word size \((16 \text{ data bits and } 6 \text{ code bits})\) we have \( k = 12 \). For 39-bits word size \((32 \text{ data bits and } 7 \text{ code bits})\) we have \( k = 14 \). For 72-bits word size \((64 \text{ data bits and } 8 \text{ code bits})\) we have \( k = 16 \). Thus, to satisfy the single-read double-fault detection property for a memory using words of 32 data bits and 7 check bits, we have to execute March SRDF1 for 14 vectors \( V_i \).

March SRDF1 was designed to meet the single-read double-fault detection property for SF, TF, WDF, RDF, DRDF, IRF, CFst or CFds faults, but no consideration of the remaining two-cell faults has been taken during its construction. Thus, in fig. 2 and 3, we introduce two slightly more lengthy algorithms (March SRDF2 and March SRDF2’), which are more convenient for covering the remaining FFMs. Before addressing the remaining FFMs, we establish in corollary 1 the validity of March SRDF2 and March SRDF2’ for SF, TF, RDF, DRDF, IRF, CFst or CFds faults. Then, we use them to design test algorithms that also cover the missed two-cell FFMs.

\[
\begin{align*}
\{& (W V_1) ; \\
\uparrow ( & R V_4, W \overline{V}_4, W V_3, W V_5, W \overline{V}_7, W \overline{V}_6, R \overline{V}_1) ; \\
& M_{11} M_{12} M_{13} M_{14} M_{15} M_{16} M_{17} ) \\
\downarrow ( & R \overline{V}_1, R \overline{V}_7, \overline{W} V_4, R V_3, \overline{W} V_5, \overline{W} V_7, \overline{W} \overline{V}_6, W V_3 ) ; \\
& M_{21} M_{22} M_{23} M_{24} M_{25} M_{26} M_{27} M_{28}
\end{align*}
\]

Figure 2. March SRDF2

\[
\begin{align*}
\{& (W V_1) ; \\
\uparrow ( & R V_4, W \overline{V}_4, W V_3, W V_5, W \overline{V}_7, W \overline{V}_6, R \overline{V}_1) ; \\
& M_{11} M_{12} M_{13} M_{14} M_{15} M_{16} M_{17} ) \\
\downarrow ( & R \overline{V}_1, R \overline{V}_7, \overline{W} V_4, R V_3, \overline{W} V_5, \overline{W} V_7, \overline{W} \overline{V}_6, W V_3 ) ; \\
& M_{21} M_{22}
\end{align*}
\]

Figure 3. March SRDF2’

**Corollary 1:** The test produced when March SRDF2 is executed for each vector \( V_i \) of a two-covering set satisfies the single-read double-fault detection property for SF, TF, WDF, RDF, DRDF, IRF, CFst or CFds faults. This is also true for March SRDF2’.

**Short proof:** Considering \( M22 \) as victim-detection read, March SRDF2 is designed similarly to SRDF1 to provide all sensitizing states/operations for the faults SF, TF, WDF, RDF, DRDF, IRF, CFst, and CFds, as well as, for the sensitizing states/operations of the two-cell faults CFst and CFds, to provide both the a = 0 and the a = 1 values of the aggressor cell. Thus, the corollary is proven similarly to proposition 1. This also holds true for SDRF2’, which is identical to SDRF2 but uses inverse addressing order. QED

Since a two-cell FFM involves a victim cell and an aggressor cell, if two two-cell FFMs affect two cells of a memory word, four cells will be involved, requiring all value combinations in four bit positions for detecting all combinations of such FFMs. Applying all value combinations in any 4-bit positions requires using a 4-bit covering set of vectors \( V_i \). Nevertheless in proposition 1 and corollary 1 we were able to cope with two important classes of two-cell FFMs (CFst and CFds) by using a two-covering set of binary vectors \( V_i \). This was possible because CFst and CFds faults are sensitized when the victim cell is in a particular state. Thus, we can employ a march element (like element M1 or element M2 in SRDF1), which performs all possible operations \((RV_i; R \overline{V}_i; V_iW \overline{V}_i; \overline{V}_i W \overline{V}_i; V_iW \overline{V}_i; \overline{V}_i W \overline{V}_i)\). These operations enable the march element to sensitise the fault for both the 0 and the 1 states of each aggressor cell, as well as for all possible operations performed over each aggressor cell (0W0, 0W1, 1W1, 1W0, R0, R1), regardless to the value of \( V_i \). As a matter of fact, we only need to use a set of binary vectors \( V_i \) that provide to the two victim cells all state combinations (00, 01, 10, 11), which can be done by using a 2-covering set of binary vectors \( V_i \). However, for the remaining two-cell faults (CFtr, CFwd, CFrd, CFird, and CFir), the situation is more difficult as stated below.

**Remark 1:** To sensitize a CFtr, CFwd, CFrd, or CFir fault the aggressor cell should have a particular value at an instant at which the victim cell undergoes a particular read or write operation. Thus, contrary to CFst and CFds, a march element cannot sensitize a CFtr, CFwd, CFrd, or CFir fault for both states of the aggressor cell, since the aggressor cell will have a precise (unique) value when the march element performs the required operation over the victim cell.

**Remark 2:** For CFtr, CFwd, CFrd, or CFir the sensitizing operations are performed over the victim cells. Thus, a march element cannot perform all sensitizing operations because a sensitizing write can masque the sensitization of a fault produced by a previous operation.

From these remarks, achieving the single-read double-fault detection property for the remaining two-cell faults is more difficult, and using a 2-covering set of vectors \( V_i \) may not be sufficient for achieving this property for all of them. In the following we analyse in more details this issue in order to
determine all faults that can be addressed by means of 2-covering sets and propose the related algorithms.

For CFtr, CFwd, CFrd, CFdrd, and CFir faults, the operation \( v \vee w \), \( v \vee \bar{w} \) or \( \bar{v} \) performed over the victim cell for sensitizing the fault is referred hereafter as victim sensitizing operation, and the associated value \( v \) as victim sensitizing state. The value \( a \) that has the aggressor cell during the victim sensitizing operation is referred as aggressor sensitizing-state. Lemma 3: The following conditions are sufficient and necessary for detecting the following FFM: \( \text{CFir} < a; v \vee \bar{w} / v / - >, \text{CFwd} < a; w \vee v / \bar{v} / - >, \text{CFrd} < a; \bar{v} \vee \bar{w} / v / - >, \text{CFdrd} < a; w \vee \bar{v} / v / - >, \text{CFir} < a; v \vee \bar{w} / \bar{v} > \) by a given read operation performed over the victim cell (victim detection read):

i.a Detection Conditions for CFtr <a; v\( w \vee \bar{v} / v / - > \): the operation \( v \vee \bar{w} \) has to be performed over the victim cell (victim-sensitizing operation) before the victim detection read; no \( w \vee \bar{w} \) write is performed over the victim cell between these operations; and no \( \bar{v} \vee \bar{w} \) write is performed over the victim cell between these operations if, in the mean time, the state of the aggressor has been changed.

i.b Detection Conditions for CFwd <a; v\( w \vee / \bar{v} / - > \): the operation \( v \vee \bar{w} \) has to be performed over the victim cell (victim-sensitizing operation) before the victim detection read, and no write is performed over the victim cell between these operations.

i.c Detection Conditions for CFrd <a; \( \bar{v} \vee / \bar{v} > \): the victim-sensitizing operation can be the victim-detection read itself or a read preceding it; and, in the later case no write operation is performed over the victim cell between these operations.

i.d Detection Conditions for CFdrd <a; \( \bar{v} \vee / v > \): the operation \( \bar{v} \vee \bar{w} \) has to be performed over the victim cell (victim-sensitizing operation) before the victim-detection read and no write is performed over the victim cell between these reads.

i.e Detection Conditions for CFir <a; \( \bar{v} \vee / v > \): the victim-sensitizing operation is (necessarily) the victim-detection read itself.

ii. In all cases: the state of the aggressor cell during the victim-sensitizing operation is equal to \( a \).

Lemma 3 can be proven trivially from the definitions of CFtr, CFwd, CFrd, CFdrd, and CFir faults. Also, we check trivially that if the conditions of Lemma 3 are realized for all value combinations 00, 01, 10 and 11 of \( a \) and \( v \), then all CFtr, CFwd, CFrd, CFdrd, or CFir faults are detected by the victim-detection read.

Remark 3: From Lemma 3, any number of reads over the victim cell can be performed between the victim-sensitizing operation and the victim-detection read.

Remark 4: Generally, a write performed over the victim after a fault-sensitizing operation will destroy the fault sensitization. However, for CFtr <a; v\( w \vee / v / - > \), if the aggressor state is \( a \) and the victim state is \( v \), performing the operation \( w \vee \) leaves the victim at the state \( v \) (which is erroneous). Thus, if the aggressor state is still \( a \) and we perform again \( w \vee \), it will again leave the victim at the state \( v \).

Hence the second \( w \vee \) does not desensitize the CFtr fault. Consequently, we can perform the two operations \( w \vee ; w \vee \), in order to both: sensitize the CFtr <a; v\( w \vee / v / - > \) with the first \( w \vee \), and sensitize CFwd <a; \( w \vee / v / - > \) with the second \( w \vee \), without affecting the sensitization of CFtr by the second \( w \vee \).

Based on remarks 1, 2, 3, 4 and lemma 3, let us determine how march test algorithms can sensitize and detect CFtr, CFwd, CFrd, CFdrd, and CFir faults.

Lemma 4:

i. A read operation (victim-detection read) can detect a CFtr, CFwd, or CFir fault for only one state of the aggressor cell.

ii. A read operation (victim-detection read) can detect a CFrd or CFdrd fault for both 0 and 1 states of the aggressor cell.

Proof of part i: In the case of CFwd faults, from condition i.b of Lemma 3, no write can be performed over the memory word containing the victim cell between the write sensitizing this fault and the read detecting it. This means that a read operation can detect a CFwd fault sensitized by a single write operation, and thus, for only one state of the aggressor cell (the state that has the aggressor during this write).

From condition i.a of Lemma 3, either no write is performed over the memory word containing the victim cell of a CFtr fault between a \( v \vee \bar{w} \) transition sensitizing this fault and its detection read (hence, as above, the fault is sensitized by a single write and thus for a single value of the aggressor cell), or a \( w \vee \bar{w} \) is performed but the state of the aggressor cell has not been changed in the mean time (hence the fault is sensitized by several writes but each time the aggressor cell has the same state). In both cases the read can detect a CFtr fault for only one state of the aggressor.

From condition i.e of Lemma 3, the sensitizing read of a CFir fault coincides with its detection read. Thus a read detects a CFir fault for only one state of the aggressor cell (i.e. for the state that has the aggressor cell during this read). QED.

Proof of part ii: From i.c and i.d, it is not forbidden to perform a read over the victim cell between a first read sensitizing a CFrd or a CFdrd faults and its detection read. Thus, after a first read sensitizing the fault and performed when the aggressor cell has a value \( a \), we can perform a write over the aggressor cell to invert its state to \( \bar{a} \) and then perform a second read to sensitize the CFrd or the CFdrd fault for the state \( \bar{a} \) of the aggressor cell. A third read will be able to detect each of these faults (e.g. the one having \( a \) as aggressor-sensitizing state and the one having \( \bar{a} \) as aggressor-sensitizing state). Note that the faults detected by this read have the same victim-sensitizing state, since no write has been performed over the victim cell between the first and the third read. Note also that for CFrd faults the second and third reads used in the above analysis can coincide. QED.

Based on lemmas 3 and 4 and their proofs, we derive the march test algorithm structure shown in Figure 4. For such algorithms we can show the following lemma.
Lemma 5. Let us consider march test algorithms comprising at least 2 march elements M1 and M2 having the following characteristics: at the beginning of M1 the state of all memory words is $V_i$; the last three operations of M1 (referred as M1a, M1b, M1c) are $W_{V_i}$, $W_{V_i}$, $R_{V_i}$; the first two operations of M2 (referred as M2a, M2b) are $R_{V_i}$, $R_{V_i}$; and the last operation of M2 (referred as M2c) is $W_{V_i}$, as shown in figure 4. Then, for such an algorithm the following properties hold true:

i. The state of the aggressor cell is inverted between the instant in which M1 visits the word containing the victim cell, and the instant in which M2 visits the word containing the victim cell, regardless to the relation of the addresses of the victim and the aggressor cells.

ii. CFtr faults are sensitized by M1a and detected in M1c as well as in M2b.\(^2\)

iii. CFwd faults are sensitized by M1b and detected in M1c and M2b.

iv. CFrd faults are sensitized by M1c and detected in M1c and M2b, and also sensitized by M2b and detected in M2b. From point i above, the aggressor-sensitizing state in M2b is the inverse of the aggressor-sensitizing state in M1c.

v. CFdrd faults are sensitized by M1c and detected in M2b and also sensitized by M2a and detected in M2b. As above, the aggressor-sensitizing state in M2a is the inverse of the aggressor-sensitizing state in M1c.

vi. CFir faults are sensitized by M1c and detected in M1c, and also sensitized by M2b and detected in M2b. The aggressor-sensitizing state in M2b is the inverse of the aggressor-sensitizing state in M1c.

Proof: Point i of lemma 5 can be proven trivially by observing that M1 inverses that states of the memory cells and so do M2. The rest of lemma 5 can be proven trivially based on lemma 3. QED

Lemma 6: The victim sensitizing state is the same for all CFtr faults detected by an algorithm obeying the structure described in lemma 5. This is also true for CFwd faults, CFrd faults, CFdrd faults and CFir faults.

Proof: This lemma can be proven trivially since any CFtr fault is sensitized by the same operation (M1a). Thus, we have a unique victim-sensitizing state for CFtr faults. The same holds true for CFwd faults, which are sensitized by operation M1b. Operations performed at several positions of the algorithm can sensitize each of the remaining faults (e.g. operations M1c, M2a and M2b for CFir). However all these operations are identical ($R_{V_i}$). Thus, all of them involve the same victim sensitizing state. QED

Lemma 7: Let us consider a double faults [f1, f2] composed of two faults f1 and f2 such that: f1 and f2 are of the types CFtr, CFwd, CFrd, CFdrd, or CFir, and f1 and f2 affect the same memory word (victim word). Let us partition these double faults into the 7 categories (i, ii, iii, iv, v, vi, and vii) according to the types of faults f1 and f2, as described in table 4. Let a1 and a2 be the values that have the aggressor cells of f1 and f2 at the instant in which the march element M1 of an algorithm obeying the structure described in lemma 5 visits the victim word. Then, for the fault [f1, f2] of each category i, ii, iii, iv, v, vi, and vii, such that both f1 and f2 are detected in M1c, or in M2a, or in M2b of this algorithm, column 4 of table 4 gives the values of the pairs of the aggressor-sensitizing states.

Table 4. Pairs of aggressor sensitizing states in algorithms obeying the structure described in lemma 5

<table>
<thead>
<tr>
<th>Fault categories</th>
<th>f1</th>
<th>f2</th>
<th>Pairs of Aggressor Sensitizing States</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>CFtr or CFwd</td>
<td>CFrd or CFdrd</td>
<td>(a1, a2), (a1, \overline{a2})</td>
</tr>
<tr>
<td>ii</td>
<td>CFtr or CFwd</td>
<td>CFtr</td>
<td>(a1, a2), (a1, \overline{a2})</td>
</tr>
<tr>
<td>iii</td>
<td>CFtr</td>
<td>CFrd</td>
<td>(a1, a2), (a1, \overline{a2}), (a2, \overline{a2})</td>
</tr>
<tr>
<td>iv</td>
<td>CFtr</td>
<td>CFdrd</td>
<td>(a1, a2), (a1, \overline{a2}), (\overline{a1}, a2)</td>
</tr>
<tr>
<td>v</td>
<td>CFdrd or CFird</td>
<td>CFrd</td>
<td>(a1, a2), (a2, \overline{a1}), (a1, \overline{a2})</td>
</tr>
<tr>
<td>vi</td>
<td>CFtr or CFwd</td>
<td>CFtr or CFrd</td>
<td>(a1, a2)</td>
</tr>
<tr>
<td>vii</td>
<td>CFtr</td>
<td>CFir</td>
<td>(a1, a2), (a1, \overline{a2})</td>
</tr>
</tbody>
</table>

Proof: To prove this lemma we need to validate the values of the aggressor sensitizing pairs reported in column 4 of table 4. This can be done trivially by using, for each FFM in the column 2 and 3 of the table, its sensitizing operations from lemma 5, and considering the value that has the aggressor cell during these operations, as detailed bellow for each of the fault categories i, ii, iii, iv, v, vi, v:

i. If f1 is a CFtr or a CFwd fault and f2 is a CFrd or a CFdrd fault, then, from points ii, iii, iv, v and vi of lemma 5, M2b detects f1 and f2 for aggressor-sensitizing states (a1, a2) as well as for aggressor-sensitizing states (a1, \overline{a2}).

ii. If f1 is a CFtr or a CFwd fault and f2 is a CFtr fault, then, from points ii, iii, and vi of lemma 5 we find that M1c detects f1 and f2 for aggressor-sensitizing states (a1, a2); and M2b detects f1 and f2 for aggressor-sensitizing states (a1, \overline{a2}).

iii. If f1 is a CFir fault and f2 is a CFrd fault, then from points iv, v, and vi of lemma 5 we find that M1c detects f1 and f2 for aggressor-sensitizing states (a1, a2); and M2b detects f1 and f2 for aggressor-sensitizing states (\overline{a1}, \overline{a2}) as well as for aggressor-sensitizing states (\overline{a1}, a2).

iv. If f1 is a CFir fault and f2 is a CFdrd fault, then, from points vi, vii, v, and vi of lemma 5 we find that M2b detects f1...
and f2 for aggressor-sensitizing states \((a_1, a_2)\), as well as for aggressor-sensitizing states \((\bar{a}_1, \bar{a}_2)\).

v. If f1 is a CFrd or a CFdrd fault and f2 is a CFrd or a CFdrd fault, then, from points iv, and v of lemma 5 we find that M2b detects f1 and f2 for aggressor-sensitizing states \((a_1, a_2)\), \((\bar{a}_1, a_2)\), and \((a_1, \bar{a}_2)\). Also, as well as for aggressor-sensitizing states \((a_1, a_2)\), \((\bar{a}_1, a_2)\), and \((a_1, \bar{a}_2)\).

vi. If f1 is a CFtr or a CFwd fault and f2 is a CFtr or a CFwd fault, then, from points i, ii, and iii of lemma 5 each of the reads M1c, M2a, and M2b detects f1 and f2 for aggressor-sensitizing states \((a_1, a_2)\).

vii. If f1 is a CFir fault and f2 is a CFir fault, then, from points vi of lemma 5, M1c detects f1 and f2 for aggressor-sensitizing states \((a_1, a_2)\), and M2a as well as M2b detects f1 and f2 for aggressor-sensitizing states \((\bar{a}_1, \bar{a}_2)\).

**Lemma 8:** Let us consider a test algorithm using the vector Vi as data background (i.e. the read and write data are Vi or \(\overline{V_i}\)). Then, for a given victim-sensitizing operation involving the aggressor-sensitizing state a and the victim-sensitizing state \(v\) and a given victim-read detection, there is a bijective relationship between \(a\) and \(Vip\) and between \(v\) and \(Vij\) (where \(Vij\) and \(Vip\) are the values that vector Vi supplies to the bit positions \(p\) and \(j\) that the aggressor and the victim cells occupy to their respective words). In particular, inverting \(Vip\) inverts \(a\) and inverting \(Vij\) inverts \(v\).

**Proof:** This lemma is proven trivially by observing that inverting the state \(Vik\) of any bit position \(k\) in vector Vi will invert all the occurrences in the test algorithm of the state of bit position \(k\) of each memory word. QED

**Proposition 2:** If we execute an algorithm obeying the structure described in lemma 5 for a 2-bit covering set of vectors Vi, then, the single-read double-fault detection property is satisfied for any double fault of category v.

**Proof:** From lemma 7, for any pair of faults f1, f2 of category v detected by the same read operation of the algorithm having the structure described in lemma 5, the pairs of aggressor sensitizing states take all four possible values 00, 01, 10 and 11 whatever are the values of a1 and a2 supplied by a test vector Vi. This is because, from column 4 in the table of lemma 7, these pairs take the values \((a_1, a_2)\), \((\bar{a}_1, a_2)\), \((a_1, \bar{a}_2)\) and \((\bar{a}_1, \bar{a}_2)\). From lemma 6, when executing the algorithm for a vector Vi, the value \(v_i\) of the victim sensitizing state of fault f1 is always the same and this is also true for the value \(v_2\) of the victim sensitizing state of fault f2. Thus, each of the pairs \((a_1, a_2)\), \((\bar{a}_1, a_2)\), \((a_1, \bar{a}_2)\) and \((\bar{a}_1, \bar{a}_2)\) is combined with the pair \((v_1, v_2)\). Then, from lemma 8 we find trivially that executing the algorithm for a 2-covering set of vectors Vi (which supplies all four possible value combinations to any two bit-positions), will provide all four possible value combinations to the victim sensitizing states of the faults f1 and f2. Thus, we obtain all 16 possible combinations for the aggressor and victim sensitizing-states for the double faults of category v.

Concerning bit positions coincidence issues, the same arguments as in the proof of proposition 1 are valid. QED

From lemma 7 (table 4), for the faults of category v, an algorithm having the structure described in lemma 5 supplies all four pairs \((a_1, a_2)\), \((\bar{a}_1, a_2)\), \((a_1, \bar{a}_2)\), \((\bar{a}_1, \bar{a}_2)\) of aggressor sensitizing-states. Thus, we can use a set of two-covered vectors to achieve the single-read double-fault detection property. Unfortunately, this is not the case for the faults of the other categories. To cope with this issue, we extend the test algorithm in the manner described in lemma 9.

**Lemma 9:** If the march elements M1 and M2 of the algorithms described in lemma 5 are replaced by two march elements M1’ and M2’, which have the same structure as M1 and M2 but use inverse address order, then, for CFtr, CFwd, CFrd, CFdrd and CFir faults, the values of the victim-sensitizing states remain unchanged but the values of the aggressor-sensitizing states are inverted.

**Proof:** The values of the victim-sensitizing states remain unchanged because the victim-detection reads M1a’, M1b’, M1c’, M2a’, M2b’, M2c’ are identical to the victim-detection reads M1a, M1b, M1c, M2a, M2b, M2c, including the read values. The values of the aggressor-sensitizing states for CFtr, CFwd, CFrd, CFdrd and CFir faults are inverted because: when we visit any memory word W1 in M1’ or M2’ to perform sensitizing operations over potential victim cells, all other memory words (hence including any word W2 that contains potential aggressor cells) have values that are inverse of the values they have when we visit W1 in M1 or M2. This fact is established trivially by noting that:

- Due to the use of inverse address order, if a word W2 is visited in march element M1’ after another word W1, then, W2 is visited in M1 before W1, and vice versa.
- The value of a word W2 after it is visited in M1’ is the inverse of its value before it is visited in M1, and vice versa.
- The above two facts hold also true for M2’ and M2.

QED

**Figure 5:** March test algorithm structure related to lemma 10.

**Lemma 10:** A march test algorithm (as the one of figure 5), comprising: two march elements M1 and M2 having the structure described in Lemma 5, and two march elements M1’ and M2’ having the similar structure but use inverse address order (as described in lemma 9), provides all the four aggressor-sensitizing states \((a_1, a_2)\), \((\bar{a}_1, a_2)\), \((a_1, \bar{a}_2)\) and \((\bar{a}_1, \bar{a}_2)\) for the fault categories i, ii, iii and iv, and the two aggressor sensitizing states \((a_1, a_2)\), \((\bar{a}_1, \bar{a}_2)\) for the fault categories vi, and vii. Furthermore, the values \(v_1\), \(v_2\) of the victim-sensitizing states are always the same.

**Proof:** The first part of this lemm comes trivially by considering the aggressor-sensitizing pairs of table 4 (from
Proposition 3: If we execute an algorithm obeying the structure described in lemma 10 for a 2-covering set of vectors \( V_i \), then, the single-read double-fault detection property is satisfied for all double faults of categories i, ii, iii, iv and v, and for the half of the faults of categories vi and vii.

Proof: For the faults of category v the proposition has been proven already (see proposition 2).

For the categories i, ii, iii, and iv: based on the results of lemma 10, the proposition is proven similarly to proposition 2 by exploiting the fact that the set of vectors \( V_i \) is 2-covering.

For the faults of categories vi and vii, the proof is done similarly by using the fact we use a 2-covering set of vectors \( V_i \) and that from lemma 10 only the half of aggressor-sensitizing states \((a_1, a_2), (\overline{a}_1, \overline{a}_2)\) are supplied for categories vi and vii.

Concerning bit positions coincidence issues, the same arguments as in the proof of proposition 1 are valid. \( \text{QED} \)

Proposition 4: If we execute the algorithm March SRDF3, (shown in figure 6) for a 2-bit covering set of vectors \( V_i \), then, the single-read double-fault detection property is satisfied for all double faults composed of single-cell FFMs and two-cell FFMs, except the half of the faults of categories vii.

Proof: Case 1. Let \( f_1 \) and \( f_2 \) be the single faults composing a double fault \([f_1, f_2]\). March SRDF3 is composed of March SRDF2 and March SRDF2'. From corollary 1, executing March SRDF2 (and also March SRDF2') for a 2-covering set of vectors \( V_i \), satisfies the single-read double-fault detection property for any double fault such that: \( f_1 \) is a single-cell FF or a two-cell FF of the types CFst and CFds and the same is true for \( f_2 \).

Case 2. We check trivially that March SRDF3 satisfies the structure shown in figure 5. Thus, from proposition 3 the single-read double-fault detection property is satisfied for all double faults such that \( f_1 \) and \( f_2 \) are two-cell FFMs of the types CFtr, CFwd, CFrd, CFdrd, and CFir, except the half of the double faults of categories vi and vii.

Case 3. To complete the proof we should address the case where \( f_1 \) is a single-cell FF or a two-cell FF of the type CFst or CFds, and \( f_2 \) is a two-cell FF of the type CFtr, CFwd, CFrd, CFdrd, or CFir.

When \( f_1 \) is a two-cell fault of type CFst or CFds, then, from the proof of corollary 1, considering M22 as detection read, March SRDF2 provides to the aggressor cell all states/operations required to sensitize such a fault. That is, both the 0 and the 1 states for CFst faults, both the r0 and r1 operations for CFds(ta), both the 0w1 and 1w0 for CFds(aw) and both the 0w0 and 1w1 for CFds(awa). Thus, two aggressors-sensitizing states \( a1 \) and \( \overline{a}_1 \) are supplied for fault \( f_1 \). This holds true also for March SRDF2' when M22 is considering as detection read. In March SRDF2, the sensitizing operations over the aggressors are performed after the victim is visited in march element M1 and before it is revisited in march element M2. During this period, the value of the victim word is \( V_i \). Thus, for each fault only one sensitized value is supplied to the victim of \( f_1 \) (say value \( v_1 \)). We check easily that during the sensitisation of \( f_1 \) in March SRDF2' the value of the victim word is again \( V_i \). Thus, for each fault \( f_1 \), the victim sensitizing value in March SRDF2' is the same as in March SRDF2 (i.e. \( v_1 \)). Therefore, for fault \( f_1 \), March SRDF2 supplies the following aggressor-sensitizing and victim-sensitizing values pairs \((a_1, v_1), (\overline{a}_1, v_1), \) and the same pairs are also supplied in March SRDF2'.

Fault \( f_2 \) should be detected by the same detection read as \( f_1 \), that is M22 in March SRDF2 or M22' in March SRDF2'. Fault \( f_2 \) is of the type CFtr, CFwd, CFrd, CFdrd, or CFir. Detecting CFtr, CFwd and CFir faults in M22 of March SRDF2, implies that the sensitizing operation performed over the victim is M15 for CFtr, M16 for CFwd, and M22 for CFir. As there is just one such operation for each of these faults, we have just one victim-sensitizing value for each of them. For CFrd and CFdrd faults we can have two sensitizing operations (M17 and M21) but as they are identical they provide a single victim-sensitizing value. Thus, we have just one victim-sensitizing value for any fault of the type CFtr, CFwd, CFrd, CFdrd, or CFir.

As operations M15', M16', M17', M21' and M22' are identical to M15, M16, M17, M21, and M22, we will have the same victim sensitizing value \( v_2 \) in March SRDF2 and March SRDF2'. On the other hand, the aggressor sensitizing state for fault \( f_2 \) takes inverse values in March SRDF2 and March SRDF2' (say \( a_2 \) and \( \overline{a}_2 \)). This gives for \( f_2 \) the sensitizing pair \((a_2, v_2)\) in March SRDF2 and \((\overline{a}_2, v_2)\) in March SRDF2'. Thus, the sensitized pairs \((a_1, v_1), (\overline{a}_1, v_1)\) of \( f_1 \) are combined in March SRDF2 with the sensitizing pair \((a_2, v_2)\) of \( f_2 \), and in March SRDF2' with the sensitizing pair \((\overline{a}_2, v_2)\) of \( f_2 \). We obtain the following sensitizing quadruples for the double fault \([f_1, f_2]\): \((a_1, v_1, a_2, v_2)\), \((\overline{a}_1, v_1, \overline{a}_2, v_2)\), \((a_1, v_1, \overline{a}_2, v_2)\), \((\overline{a}_1, v_1, a_2, v_2)\). Then, executing March SRDF3 for a 2-covering set of vectors \( V_i \) will provide all 4 values to the victim sensitizing states \( v_1 \) and \( v_2 \). In this case we find trivially that, the above 4 quadruplets provide all the 16 possible sensitizing quadruples for the double fault \([f_1, f_2]\). This guarantees that each pair of faults \( f_1 \) and \( f_2 \) is detected by the operation M22 or the operation M22' in some of the executions of SRDF3.

Above we considered that \( f_1 \) is a two-cell fault of the type CFst or CFds. When \( f_1 \) is a single-cell FF (i.e. there is no aggressor cell), in the previous arguments we eliminate the aggressor cell of \( f_1 \) and we obtain the proof for this case too.

Furthermore, as the vectors \( V_i \) were used only for applying all possible values to the victim sensitizing states \( v_1 \) and \( v_2 \), using similar arguments as in proposition 1 implies that bit positions coincidences do not affect the arguments used in this proof. \( \text{QED} \)
includes a read double fault detection property for the large majority of them. Indeed, the single-read double-fault detection property is not satisfied only for 4 of these combinations (corresponding to the categories vi and vii of table 4), and only for the half of the faults of these categories.

So, the coverage achieved by March SRDF3 could be considered satisfactory. However, if a higher fault coverage is required, we need to cope with the double faults of categories vi and vii. That is, when both faults are of the type CFtr; or both faults are of the type CFwd; or the one fault is CFtr and the other is CFwd. The coverage of these faults is discussed next. First we show that covering these faults requires using a 4-covering set of vectors $\mathcal{V}_i$.

**Lemma 11:** For any given double fault $[f_1, f_2]$ let $a_1, a_2, v_1, v_2$ be the values of the bit positions of a vector $\mathcal{V}_i$ corresponding to the bit positions of the aggressor cells and victim cells of this fault. A test algorithm that uses the vector $\mathcal{V}_i$ and its inverse $\overline{\mathcal{V}}_i$ as data in any read and write operation, could achieve the single-read double-fault detection property for the double faults of categories vi and vii for the sensitizing quadruplets $(a_1, a_2, v_1, v_2), (\overline{a_1}, \overline{a_2}, v_1, v_2), (a_1, a_2, \overline{v_1}, \overline{v_2}), (\overline{a_1}, \overline{a_2}, \overline{v_1}, \overline{v_2})$, and only for these quadruplets.

**Proof:** We can easily check that the algorithm shown in figure 7 achieves the single-read double-fault detecting property for the double faults of categories vi and vii corresponding to the sensitizing quadruplets: $(a_1, a_2, v_1, v_2), (a_1, a_2, \overline{v_1}, \overline{v_2}), (\overline{a_1}, \overline{a_2}, v_1, v_2), (\overline{a_1}, \overline{a_2}, \overline{v_1}, \overline{v_2})$, and $(a_1, a_2, \overline{v_1}, \overline{v_2}), (\overline{a_1}, \overline{a_2}, v_1, v_2)$. This proves the 1st part of the lemma.

Concerning the 2nd part (only faults corresponding to the quadruplets $(a_1, a_2, v_1, v_2), (a_1, a_2, \overline{v_1}, \overline{v_2}), (\overline{a_1}, \overline{a_2}, v_1, v_2), (\overline{a_1}, \overline{a_2}, \overline{v_1}, \overline{v_2})$ can be detected), we first note that a test algorithm using $\mathcal{V}_i$ and $\overline{\mathcal{V}}_i$ as data in its read and write operations can include only the following four operations: $\mathcal{V}_i, W \mathcal{V}_i, R \mathcal{V}_i$. Let i, j, p and q be the bit positions of the aggressor and victim cells of a double fault $[f_1, f_2]$. Let $a_1, a_2, v_1$ and $v_2$ be the values of the bit positions i, j, p and q in vector $\mathcal{V}_i$. Let us consider a test algorithm that includes a read detecting both $f_1$ and $f_2$ (detection read). Let both $f_1$ and $f_2$ be CFtr faults. From the detection conditions of CFtr faults given earlier (lemma 3), we have that: between the operation $\mathcal{V}_i W \mathcal{V}_i$ sensitizing a CFtr fault and the read operation detecting this fault no $\mathcal{V}_i W \mathcal{V}_i$ operation can be performed over the victim cell. This implies that if a CFtr fault is sensitized by the operation $\mathcal{V}_i W \mathcal{V}_i$ and is detected by a read, then a CFtr fault sensitized by the operation $\overline{\mathcal{V}}_i W \mathcal{V}_i$ can not be detected by the same read, and vice-versa. Thus, as $f_1$ and $f_2$ affect the same memory word, if they are detected by the same read they are sensitized by the same operation, which is either $\mathcal{V}_i W \mathcal{V}_i$ or $\overline{\mathcal{V}}_i W \mathcal{V}_i$. As a consequence, the possible combinations of victim sensitizing states are $(v_1, v_2)$ and $(\overline{v_1}, \overline{v_2})$. Also, as the content of a memory word is either vector $\mathcal{V}_i$ or its inverse (i.e. $\overline{\mathcal{V}}_i$), at each given instant of the test algorithm the values of the aggressor cells are either $a_1$ and $a_2$, or $\overline{a_1}$ and $\overline{a_2}$. As $f_1$ and $f_2$ are sensitized by the same operation, the aggressor sensitizing states will be the values that have the aggressor cells during this operation, yielding to two possible aggressor-sensitizing states combinations: $(a_1, a_2)$ and $(\overline{a_1}, \overline{a_2})$. Thus, the test algorithm could provide only the following combinations of sensitizing quadruplets: $(a_1, a_2, v_1, v_2), (a_1, a_2, \overline{v_1}, \overline{v_2}), (\overline{a_1}, \overline{a_2}, v_1, v_2), (\overline{a_1}, \overline{a_2}, \overline{v_1}, \overline{v_2})$.

The proof is given in similar manner when $f_1$ and $f_2$ are both CFwd faults, as well as when $f_1$ and $f_2$ are both CFtr faults. When the one fault is CFtr and the other is CFwd, the proof can also be done with similar arguments, while paying attention to the fact that the remark 4 related to lemma 3 is not helpful for increasing the victim sensitizing values:

i. the detection condition for CFtr < $a$; $\mathcal{V}_i W \mathcal{V}_i$ / $v$ / $\gg$ > allows performing any number of $\mathcal{V}_i W \mathcal{V}_i$ operations between the victim sensitizing operation $\mathcal{V}_i W \mathcal{V}_i$ and the detection read;

ii. however this is allowed only if the state of the aggressor is the same during the victim sensitizing operation $\mathcal{V}_i W \mathcal{V}_i$ and the subsequent $\mathcal{V}_i W \mathcal{V}_i$ operations.

Thus, while point i. seems allowing performing the sensitizing operations of the CFtr and the CFwd faults at different instants of the algorithm, which could be exploited for modifying the values of the aggressor cells between the sensitization operation of $f_1$ and that of $f_2$ (introducing this way the aggressor sensitizing pair $(a_1, a_2)$), point ii prevents us from doing it. QED

\[\{\mathcal{G}(W \mathcal{V}_i); \hspace{1cm} \mathcal{G}(R \mathcal{V}_i); \hspace{1cm} \mathcal{G}(W \overline{\mathcal{V}}_i); \hspace{1cm} \mathcal{G}(W \overline{\mathcal{V}}_i); \hspace{1cm} \mathcal{G}(W \mathcal{V}_i); \hspace{1cm} \mathcal{G}(W \overline{\mathcal{V}}_i); \hspace{1cm} \mathcal{G}(W \mathcal{V}_i); \hspace{1cm} \mathcal{G}(W \overline{\mathcal{V}}_i)\}\]

**Figure 7.** Algorithm structure related to lemma 11
Lemma 12: Executing march test algorithms for a 2-covering set or a 3-covering set of vectors $V_i$ does not guaranty satisfying the single-read double-fault detection property for all faults of categories vi and vii.

Proof: Let us consider any march test algorithm using a vector $V_i$ as data background. Such an algorithm can perform read and write operations using $V_i$ and $\overline{V_i}$ as data. Thus, the conditions of lemma 11 are satisfied. From this lemma, we have that march test algorithms using a vector $V_i$ as data background can allow satisfying the single-read double-fault detection property for the faults of categories vi and vii corresponding to the sensitizing quadruplets $(a_1, a_2, v_1, v_2)$, $(a_1, a_2, \overline{v_1}, \overline{v_2})$, $(a_1, a_2, v_1, v_2)$, and $(a_1, a_2, \overline{v_1}, \overline{v_2})$ and only to these quadruplets. Supplying the variables $a_1, a_2, v_1, v_2$ of these quadruplets with the values of a 2-covering set does not guaranty producing all possible 16 values combinations. This can be shown easily by example. For instance, if we give to the variables $a_1, a_2, v_1, v_2$ of the quadruplets $(a_1, a_2, v_1, v_2) (a_1, a_2, \overline{v_1}, \overline{v_2})$, $(a_1, a_2, v_1, v_2)$, and $(a_1, a_2, \overline{v_1}, \overline{v_2})$ the values of the form of the following 2-covering set of 4-bit vectors $0000$, $0011$, $1111$, $1100$, $1010$, we obtain the vectors $0000$, $0011$, $1011$, $1100$, $1010$, $1010$, $1100$, $1111$, which form a 3-covering set, we obtain the same set of 8 quadruplet values, which are the half of the 16 possible quadruplet values.

The case concerning the 3-covering sets can also be proved by example. For instance, if we give to the variables $a_1, a_2, v_1, v_2$ of the quadruplets $(a_1, a_2, v_1, v_2) (a_1, a_2, \overline{v_1}, \overline{v_2})$, $(a_1, a_2, v_1, v_2)$, and $(a_1, a_2, \overline{v_1}, \overline{v_2})$ the values of the following 8 quadruplet values $0000$, $0011$, $0110$, $1100$, $1010$, $1100$, $1111$, which form a 3-covering set, we obtain the same set of 8 quadruplet values, which are the half of the 16 possible quadruplet values.

Thus, executing march tests for a 2-covering or a 3-covering set of vectors does not guaranty supplying all the 16 sensitizing quadruplets for the faults of categories vi and vii.

QED

From lemma 12, 2-covering and 3-covering sets do not guaranty providing the necessary values for supplying all the 16 sensitizing quadruplets for the faults of categories vi and vii. We can supply these values by providing on $a_1, a_2, v_1, v_2$ all 16 quadruplet values by means of a 4-covering set of vectors $V_i$. On the other hand, this is also possible by providing only 4 selected quadruplet values on $a_1, a_2, v_1, v_2$. For instance, if we provide on $a_1, a_2, v_1, v_2$ the 4 quadruplet values $0000$, $0001$, $0100$, and $0010$, the quadruplets $(a_1, a_2, v_1, v_2)$, $(a_1, a_2, \overline{v_1}, \overline{v_2})$, $(a_1, a_2, v_1, v_2)$, $(a_1, a_2, \overline{v_1}, \overline{v_2})$ give all the 16 possible quadruplet values. We obtain the same result for any four quadruplet values obtained by inverting the two first bits, or the two last bits, or all four bits in any of the four quadruplet values $0000$, $0001$, $0100$, $0010$. This gives 256 sets of four quadruplet values, each of whom gives the desired result. Then, we need a set of vectors $V_i$ that provides at each four bits the values of one of these 256 sets of 4 quadruplets. This can reduce the number of vectors $V_i$ with respect to a 4-covering set. However, as there are no such sets of vectors in the literature, their creation requires intensive simulated annealing computations, as done for creating 4-covering sets [29][30], which have to be done. Thus, in this paper we use a 4-covering set of vectors $V_i$, which is lengthy. So, for moderating the test length, we developed a short test algorithm that can be combined with the 4-covering set.

Proposition 5: Executing March SRDF4 (shown in figure 8) for a 4-covering set of vectors $V_i$ satisfies the single read double-fault detection property for the double faults $[f_1, f_2]$ of categories vi and vii, provided that there is no coincidence of the bit position of an aggressor cell with the bit positions of the other aggressor cell or with the bit position of a victim cell.

Proof: Faults of category vi. In March SRDF4, operation M11 sensitizes the CFtr fault $<a; v v' v'/v >$, where $v$ is the value supplied by vector $V_i$ to the bit position of the victim cell; $a$ is the value supplied by $V_i$ to the bit position of the aggressor cell when $a > v$, or its inverse when $a < v$ (where $a$ is the address of the aggressor cell and $v$ the address of the victim cell). Then, operation M13 detects the CFtr fault for these values of $v$ and $a$.

M12 sensitizes the CFwd fault $<a; v v' v'/v >$, where $v$ is the value supplied by $V_i$ to the bit position of the victim cell; $a$ is the value supplied by $V_i$ to the bit position of the aggressor cell when $a > v$, or its inverse when $a < v$. Then, operation M13 detects the sensitized CFwd fault for these values of $v$ and $a$.

From the above, if $f_1$ and $f_2$ are CFtr or CFwd faults (double-fault of category vi), M13 will detect the double fault corresponding to the sensitizing quadruplet $(a_1, a_2, v_1, v_2)$ if $a > v$, or $(\overline{a_1}, \overline{a_2}, v_1, v_2)$ if $a < v$, determined by four different bit positions of $V_i$ (since according to the statement of the proposition, there are no bit position coincidences for the aggressor and the victim cells). Thus, executing March SRDF4 for each vector $V_i$ of a 4-covering set will produce all 16 value combinations to both the quadruplets $(a_1, a_2, v_1, v_2)$ and $(\overline{a_1}, \overline{a_2}, v_1, v_2)$. Thus, each of the 16 possible double faults of category vi will be detected by operation M13 in at least one of these executions.

Faults of category vii. M13 sensitizes and detects the CFir fault. If $f_1$ and $f_2$ are CFir faults (double-fault of category vii), we find similarly a quadruplet $(a_1, a_2, v_1, v_2)$ determined by four different bit positions of vector $V_i$. Thus, executing
March SRDF4 for each vector Vi of a 4-covering set, guarantees that each of the 16 possible double faults of category vii will be detected by operation M13 in at least one of these executions. **QED**

$$
\begin{align*}
&\{Wv_1\}; \\
&\{Wf_1, Wf_2, Rf_2\}; \\
&M11 \quad M12 \quad M13
\end{align*}
$$

**Figure 8:** March SRDF4

In the following we discuss the case of faults not covered by proposition 5 due to bit positions coincidences.

**Proposition 6:** Executing March SRDF5 (shown in figure 9) for a 3-covering set of vectors Vi achieves the single-read double-fault detecting property for double faults of categories vi and vii for which the bit position of the one aggressor cell coincides with the bit position of the one victim cell and the bit position of the other aggressor cell is different from the bit position of both victim cells.

**Proof:** We find easily that the March SRDF5 shown in figure 9 achieves the single-read double-fault detection property for the double faults of categories vi and vii corresponding to sensitizing quadruplets \((a1, a2, v1, v2)\), and \((a1, a2, \overline{v1}, \overline{v2})\).

If the bit position of the aggressor of f2 coincides with the bit position of the victim of f2, then, the above sensitizing quadruplets become \((a1, a2, v1, a2)\), and \((a1, a2, \overline{v1}, a2)\), where \(a1\), and \(v1\) are the values supplied by vector Vi on the bit positions of the aggressor and victim cells of fault f1, and \(a2\) is the value supplied by Vi on the common bit position of the aggressor and victim cells of fault f1. Using a 3-covering set of vectors Vi will produce the set of all 8 possible value combinations on \((a1, a2, v1)\), as well as all 8 possible value combinations on \((a1, a2, \overline{v1})\). Then we can check trivially that the values supplied on \((a1, a2, v1, a2)\), and \((a1, a2, \overline{v1}, a2)\) cover all 16 possible quadruplet values.

Obviously, the similar arguments hold when the bit position of the aggressor of f2 coincides with the bit position of the victim of f1, and as when the bit position of the aggressor of f1 coincides with the bit position of the victim of f1 or with the bit position of the victim of f2. **QED**

\[
\begin{align*}
&\{Wv_1\}; \quad (Wf_1, Wf_2, Rf_2); \\
&\{Wv_1, Wf_2, Rf_1\};
\end{align*}
\]

**Figure 9:** March SRDF5

A particular case of faults of categories vi and vii is the case where the aggressor cell of f1 coincides with the aggressor cell of f2. For some of these faults no test algorithm can achieve the single-read double-fault detection property. Surprisingly, as shown in the next two lemmas, march test algorithms allow coping with these faults in the context of the present study.

**Lemma 13:** Executing March SRDF4 for a 3-covering set of vectors Vi and March SRDF5 for a 2-covering set of vectors Vi guaranties the single-read double-fault detection property for the double faults \([f1, f2]\) of categories vi and vii in which f1 and f2 share the same aggressor cell (common aggressor) and in which the aggressor sensitizing value of f1 is equal to the aggressor sensitizing value of f2.

**Proof:** Let us first consider the case where the two aggressor cells have the same bit position but this bit position is different from the bit position of each of the victim cells. Then, March SRDF4 supplies the sensitizing quadruplets \((a1, a1, v1, v2)\). Executing it with a 3-covering set of vectors Vi supplies all 8 quadruplets in which the sensitizing values of the two aggressors are equal.

Let us also consider that the common bit position of the aggressor cells coincides with the bit position of one of the victim cells (e.g. with the bit position of the victim of f1), then, we will have \(a1 = a2 = v1\) and March SRDF5 supplies the sensitizing quadruplets \((a1, a1, v1, v2)\). It will also supply \((a1, a1, \overline{a1}, \overline{v2})\) as we have seen in the proof of proposition 6.

We check trivially that executing March SRDF5 for a 2-covering set of vectors Vi supplies all 8 quadruplets in which the sensitizing values of the two aggressors are equal. **QED**

**Lemma 14:** No test algorithm can ensure the single-read double-fault detection property for the faults of categories vi and vii in which f1 and f2 share the same aggressor cell (common aggressor) and in which the aggressor sensitizing value of f1 is the inverse of the aggressor sensitizing value of f2, but these faults are meaningless in the context of the present study because they never produce double errors.

**Proof:** Let us first consider the case of a fault of category vi (i.e. f1 is CFlr or CFwd and f2 is CFtr or CFwd). Because f1 and f2 are sensitized by different states of their common aggressor cell, then, it is not possible to sensitize both of them by the same write performed over the word comprising the two victim cells. Then, let us suppose that during system operation or during test, one of the faults is sensitized. To sensitize the second fault another write has to be performed with the common aggressor cell being at a different state. As for this state the first fault is not sensitized, the new value written by the second write over the victim cell of this fault will bring this cell in a correct state. Thus, at any time only one cell can have erroneous value, implying that double errors cannot be produced during test, nor during system operation.

Let us now consider the case where the fault is of category vii (i.e. f1 and f2 are both CFir). Because f1 and f2 are sensitized by different states of the common aggressor cell, then, it is not possible to sensitize both of them by the same write performed over the word comprising the two victim cells. Furthermore, each time one of these faults is sensitized, the victim cell provides an erroneous read value (single-error) but its state remains correct. Then, any subsequent read could again sensitize only one of the two faults, producing again a single-error. It results that the two victim cells can never provide erroneous values in the same read.

Consequently, in both cases (fault of category vi sensitized by opposite states of the common aggressor cell, or fault of category vii sensitized by opposite states of the common aggressor cell), the occurrence of a double error is impossible during test as well as during system operation. Thus, on the one hand there is no test algorithm able to achieve the single-read double-fault detection property for these faults, but on the other hand, we do not need to repair the word containing...
such a fault because it never produces double errors. Therefore, we do not need to achieve the single-read double-fault detection property for these faults. QED.

Proposition 7 provides the tests enabling covering all faults of categories vi and vii treated so far and few other cases.

**Proposition 7:** Executing March SRDF4 for a 4-covering set of vectors Vi and March SRDF5 for a 3-covering set of vectors Vi, achieves the single-read double-fault detection property for all faults of categories vi and vii excepting half of the faults of the following two cases:

a. The bit positions of the two aggressor cells coincide (but the aggressor cells themselves do not coincide).

b. The bit position of the one aggressor cell coincides with the bit position of one victim cell and the bit position of the other aggressor cell coincides with the bit position of the other victim cell.

**Proof:** From proposition 5, we cover all faults of categories vi and vii for which there is no coincidence of the bit position of the one aggressor cell with the bit positions of the other aggressor cell or with the bit position of a victim cell. From proposition 6 we cover the cases where the bit position of the one aggressor cell coincides with the bit position of the victim cell and the bit position of the other aggressor cell is different from the bit position of both victim cells.

From lemmas 13 and 14 we cover the case where the two faults share the same aggressor cell.

Remind that the case where the two victim cells coincide is not of interest in this study (because in this case no double error is produced as the errors occur in the same cell). Note also that if the victim cell of a fault coincides with the aggressor cell of another fault, then, the sensitization of the first fault modifies the state of its victim cell, which is also the state of the aggressor of the second fault. Thus, the sensitization of the first fault can mask the second fault, meaning that the faults are linked. Therefore, as the FFMs of tables 1 and 2 are unlinked, we do not need to consider the case where the aggressor cell of a fault coincides with the victim cell of another fault.

Thus, the only cases of faults of categories vi and vii not yet treated are the following:

a. The bit positions of the two aggressor cells coincide (but the aggressor cells themselves do not coincide).

b. The bit position of the one aggressor cell coincides with the bit position of one victim cell, but these cells do not coincide; and the bit position of the other aggressor cell coincides with the bit position of the other victim cell, but these cells do not coincide.

In the proof of proposition 6 we found that March SRDF5 achieves the single-read double-fault detection property for the double faults of categories vi and vii corresponding to the sensitizing quadruplets (a1, a2, v1, v2), and (a1, a2, v1, v2).

**Case a:** in this case the above quadruplets become (a1, a1, v1, v2), and (a1, a2, v1, v2). As March SRDF5 is executed for a 3-covering set of vectors Vi, all 8 values combinations are supplied on a1, v1, and v2. Then, (a1, a1, v1, v2) gives 8 sensitizing quadruplets, and (a1, a2, v1, v2) gives the same quadruplets (all 8 quadruplets in which the sensitizing values of the two aggressor cells are equal). As all possible sensitized quadruplets are 16, the single-read double-fault detection property is satisfied for the half of the faults of case a).

In case b we have two possible situations:

b.1. a1 = v1 and a2 = v2; giving the sensitizing quadruplets (a1, a2, a1, a2), and (a1, a2, a1, a2).

b.2. a1 = v2 and a2 = v1; giving the sensitizing quadruplets (a1, a2, a1), and (a1, a2, a1, a1).

**Case b:** as March SRDF5 is executed for a 3-covering set of vectors Vi, all 4 value combinations are supplied on a1 and a2. Then, (a1, a2, a1, a2), and (a1, a2, a1, a1) give 8 sensitizing quadruplets: all 4 quadruplets in which the sensitizing states of the aggressors of f1 and f2 are equal respectively to the sensitizing states of the victims of f1 and f2; and all 4 quadruplets in which the sensitizing states of the aggressors of f1 and f2 are equal respectively to the inverse of the sensitizing states of the victims of f1 and f2.

**Case b.ii:** As all 4 value combinations are supplied on a1 and a2 we obtain 8 sensitizing quadruplets: all 4 quadruplets in which the sensitizing states of the aggressors of f1 and f2 are equal respectively to the sensitizing states of the victims of f2 and f1; and all 4 quadruplets in which the sensitizing states of the aggressors of f1 and f2 are equal respectively to the inverse of the sensitizing states of the victims of f2 and f1.

Thus, the single-read double-fault detection property is also satisfied for the half of the faults in the cases b.i and b.ii.

**QED**

From propositions 4 and 7, we find that executing March SRDF3 for a 2-bit covering set of vectors Vi; March SRDF4 for a 4-covering set of vectors Vi; and March SRDF5 for a 3-covering set of vectors Vi achieves the single-read double-fault detection property for all double-faults composed of single-cell FFMS and/or double-cell FFMS (tables 1 and 2), except a very small subset of the faults of categories vi and vii specified in proposition 7 and its proof. Lemma 15 shows that these faults cannot be covered by march test algorithms and will be referred as march-SRDF-deceptive faults.

**Lemma 15.** For the faults of categories vi and vii that are not covered in proposition 7, it is impossible to satisfy the single-read double-fault detection property by means of march test algorithms.

**Proof.** The read and write operations in any march test algorithm use Vi and V' as data (where Vi is the vector used as data background). Thus, the conditions of lemma 11 are satisfied. Hence, based on this lemma, a march test algorithm can achieve the single-read double-fault detection property for the double faults of categories vi and vii corresponding to the sensitizing quadruplets (a1, a2, v1, v2), (a1, a2, v1, v2), (a1, a2, v1, v2), and (a1, a2, v1, v2), and only to these quadruplets (where a1, a2, v1, v2 are the values that vector Vi supplies to the bit positions of the two aggressor and the two victim cells). Thus, for quadruplets different from the above (the only part of lemma 11), no march test algorithm can satisfy the single-read double-fault detection property.

For the faults of case a, of proposition 7, the bit positions of the two aggressor cells coincide. Thus, the above quadruplets become (a1, a1, v1, v2), (a1, a1, v1, v2), (a1, a1, v1, v2), and only to these quadruplets (where a1, a2, v1, v2 are the values that vector Vi supplies to the bit positions of the two aggressor and the two victim cells).
and \((a_1, \overline{a_1}, \overline{a_2}, v_2)\). If we set a set of vectors \(V_i\) supplying all possible 8 value combinations on the variables \(a_1, v_1\) and \(v_2\), we check easily that each of these quadruplets gives the same 8 value combinations (all the 8 quadruplets in which the sensitizing values of the two aggressor cells are equal). These quadruplets are the same as those covered in case a. of the proof of proposition 7. As the single-read double-fault detection property cannot be satisfied for the remaining 8 quadruplets (the “only” part of lemma 11), proposition 7 is shown for the faults of case a.

For the faults of case b.i (see proof of proposition 7), we set \(a_1 = v_1\) and \(a_2 = v_2\). In this case the four quadruplets of lemma 11 become \((a_1, a_2, a_1, a_2), (a_1, a_2, \overline{a_2}, a_1), (\overline{a_1}, \overline{a_2}, a_2, a_1),\) and \((\overline{a_1}, \overline{a_2}, \overline{a_1}, \overline{a_2})\). Using a set of vectors \(V_i\) supplying all possible 4 value combinations on the variables \(a_1\) and \(a_2\) gives 8 sensitizing quadruplets: all 4 quadruplets in which the sensitizing states of the aggressors of \(f_1\) and \(f_2\) are equal respectively to the sensitizing states of the victims of \(f_1\) and \(f_2\); and all 4 quadruplets in which the sensitizing states of the aggressors of \(f_1\) and \(f_2\) are equal respectively to the inverse of the sensitizing states of the victims of \(f_1\) and \(f_2\), which are the same as those covered in case b.i. of proposition 7. As the single-read double-fault detection property cannot be satisfied for the remaining 8 quadruplets (the “only” part of lemma 11), proposition 7 is shown for the faults of case b.i.

For the faults of case b.ii (see proof of proposition 7), we set \(a_1 = v_2\) and \(a_2 = v_1\). In this case four quadruplets of lemma 11 become \((a_1, a_2, a_2, a_1), (a_1, a_2, \overline{a_2}, a_1), (\overline{a_1}, \overline{a_2}, a_2, a_1),\) and \((\overline{a_1}, \overline{a_2}, \overline{a_1}, \overline{a_2})\). Using a set of vectors \(V_i\) supplying all possible 4 value combinations on the variables \(a_1\) and \(a_2\) gives 8 sensitizing quadruplets: all 4 quadruplets in which the sensitizing states of the aggressors of \(f_1\) and \(f_2\) are equal respectively to the sensitizing states of the victims of \(f_2\) and \(f_1\); and all 4 quadruplets in which the sensitizing states of the aggressors of \(f_1\) and \(f_2\) are equal respectively to the inverse of the sensitizing states of the victims of \(f_2\) and \(f_1\), which are the same as those covered in case b.ii of proposition 7. As the single-read double-fault detection property cannot be satisfied for the remaining 8 quadruplets (the “only” part of lemma 11), proposition 7 is shown for the faults of case b.ii. QED

From Lemma 15, the fault cases that are not covered by the proposed march test algorithms cannot be covered by any march tests. The case of these faults is further discussed in subsection IV.D.

**C. Faults of multiplicity higher than 2**

**Lemma 16:** Executing March SRDF3 for a 2-bit covering set of vectors \(V_i\), guarantees achieving the single-read double-fault detection property for all faults of multiplicity higher than 2, which are not composed exclusively of FFMs of the types CFtr and CFwd nor exclusively of FFMs of the type Fir.

**Proof:** If a multiple fault is not composed exclusively of FFMs of the types CFtr and CFwd, or exclusively of FFMs of the type Fir, then, there are at least two single faults \(f_i\) and \(f_j\) belonging to the single faults composing the multiple fault, such that the double fault \([f_i, f_j]\) is neither of category vi nor of category vii. Thus, proposition 4 guaranties the detection of \([f_i, f_j]\) and lemma 2 the detection of the multiple fault. QED

As we have shown that we cannot achieve the single-read double-fault detection property for all the faults of multiplicity 2 (lemma 15) by means of march test algorithms, one can consider that this is also the case for faults of multiplicity higher than 2. Surprisingly, the following proposition shows that the march test algorithms proposed in the previous sections cover all faults of multiplicity higher than 2. The reason for this outcome is that faults of multiplicity higher than 2 comprise several double faults, and, as we show in the following proposition, it is not possible for all of them to satisfy the conditions that classify them as of march-SRDF-deceptive faults. Due to lemma 2, this implies that all the faults of multiplicity higher than 2 are covered by march tests, and in particular by our march tests that cover all double faults except the march-SRDF-deceptive ones.

**Proposition 8:** Executing March SRDF3 for a 2-bit covering set of vectors \(V_i\), March SRDF4 for a 4-covering set of vectors \(V_i\), and March SRDF5 for a 3-covering set of vectors \(V_i\), achieves the single-read double-fault detection property for all faults of multiplicity higher than 2.

**Proof:** We will represent the bit positions of the aggressor and the victim cell of a fault \(f_i\) as \(ap(f_i)\) and \(vp(f_i)\).

Let us consider a memory word \(W\) in which \(k\) cells \(c_1, c_2, \ldots, c_k\) are affected by faults.

Thanks to lemma 2 it is sufficient to prove the proposition for multiple faults \([f_1, f_2, \ldots, f_k]\) affecting \(k\) distinct cells of the same memory word (with \(k \geq 3\)), such that each of the faults \(f_1, f_2, \ldots, f_k\) is a single FFM. Furthermore, thanks to the same lemma, for a fault \([f_1, f_2, \ldots, f_k]\) it is sufficient to prove the proposition just for two of the faults \(f_1, f_2, \ldots, f_k\). Let \(f_i, f_j\) be such two faults. From the above: \(f_i\) is a single FFM; \(f_j\) is a single FFM; \(f_i\) and \(f_j\) affect distinct victim cells. As the victim cells of \(f_i\) and \(f_j\) are distinct and belong to the same memory word, they have different bit positions. This condition can be written as: \(vp(f_i) \neq vp(f_j)\) (1)

Thanks to lemma 16 we only need to prove the proposition 8 for: the multiple faults that are composed exclusively of FFMs of the types CFtr and CFwd (in which case, all the double faults comprised in the multiple fault will belong to the category vi); and the multiple faults that are composed exclusively of FFMs of the type Fir (in which case, all the double faults comprised in the multiple fault will belong to the category vii).

In proposition 7, for a pair of faults \([f_i, f_j]\) we can:
- write condition a as:
  a) \(ap(f_i) = ap(f_j)\),
- split condition b in two cases written as:
  b1) \(ap(f_i) = vp(f_i)\) and \(ap(f_j) = vp(f_j)\)
  b2) \(ap(f_i) = vp(f_j)\) and \(ap(f_j) = vp(f_i)\).

From proposition 7, the single-read double-fault detection property is guaranteed to be satisfied for \([f_i, f_j]\) if \([f_i, f_j]\) does not satisfy condition a, neither condition b1, nor condition b2. Thus, it is sufficient to show that, for at least one double fault comprised in a multiple fault \([f_1, f_2, \ldots, f_k]\), none of these conditions is satisfied.

Let us consider the following conditions concerning the bit positions of the aggressor and victim cells of \([f_1, f_2, \ldots, f_k]\):
c.1) The aggressor cell of each fault has the same bit position as the victim cell of the fault.

c.2) All aggressor cells have the same bit position.

There are three possible cases concerning the validity of these conditions: both c.1 and c.2 are false, which can be represented as \textbf{c.1 and c.2}, c.1 is true, which can be represented as \textbf{c.1}, c.2 is true, which can be represented as \textbf{c.2}. Below we discuss each of them.

\[ \text{[f1, f2 ... fk] satisfies c.1 and c.2:} \]

In this case we find trivially that the following two conditions hold:

\textbf{c.1:} For at least one of the faults f1, f2 ... fk the bit position of its aggressor cell is different from the bit position of its victim cell. Let f1 be such a fault.

\textbf{c.2:} Since c.2 is false, the bit position of the aggressor cell of f1 cannot be equal with the bit positions of the aggressor cells of all other faults f2, ... fk. Thus, for at least one of the faults f2 ... fk (e.g. for f2) \textbf{c.2} implies ap(f1) \neq ap(f2).

Thus, conditions \textbf{c.1} and \textbf{c.2} can be written as:

\begin{align*}
\text{c.1:} & \quad \text{ap(f1) \neq vp(f1)} \\
\text{c.2:} & \quad \text{ap(f1) \neq ap(f2).}
\end{align*}

For the double fault [f1, f2], \textbf{c.2} excludes the condition a) of proposition 7 and \textbf{c.1} excludes condition b1). Thus, in the case \textbf{c.1 and c.2 we just need to exclude condition b2}.)

There are two possibilities concerning ap(f1) and vp(f2):

- either ap(f1) \neq vp(f2) \quad (2)
- or ap(f1) = vp(f2) \quad (3)

If ap(f1) \neq vp(f2) condition b2) is excluded for [f1, f2]. Thus, proposition 8 is proven for case (2).

Hence, we only need to treat the case (3). This can be combined with two possible cases concerning ap(f2) and vp(f1): ap(f2) \neq vp(f1) and ap(f2) = vp(f1), giving:

- ap(f1) = vp(f2) and ap(f2) \neq vp(f1) \quad (3.1)
- ap(f1) = vp(f2) and ap(f2) = vp(f1) \quad (3.2)

The relation ap(f2) \neq vp(f1) in (3.1) excludes b2 for the fault [f1, f2]. Thus, proposition 8 holds true for case (3.1). Hence, we only need to treat case (3.2). Since we analyze faults of multiplicity higher than 2, we can consider a third fault in the list f1, f2 ... fk (e.g. fk). Then, (3.2) can be combined with two possible cases concerning ap(f1) and ap(fk): ap(f1) = ap(fk) and ap(f1) \neq ap(fk), giving:

\begin{align*}
ap(f1) = vp(f2), \quad ap(f2) = vp(f1), \quad ap(f1) = ap(fk) & \quad (3.2.1) \\
ap(f1) = vp(f2), \quad ap(f2) = vp(f1), \quad ap(f1) \neq ap(fk) & \quad (3.2.2)
\end{align*}

In case of (3.2.1), considering the relation ap(f1) \neq ap(f2) from condition \textbf{c.2} gives ap(f2) \neq ap(fk), which excludes condition a) of proposition 7 for the fault [f2, fk]. From (1) we obtain vp(f1) \neq vp(f2) and vp(f1) \neq vp(fk). Thus, ap(f2) = vp(f1) in (3.2.1) gives ap(f2) \neq vp(f2) and ap(f2) \neq vp(fk). For the fault [f2, fk], these relations exclude the conditions b1) and b2).

In case (3.2.2), ap(f1) \neq ap(fk) invalidates condition a) of proposition 7 for the fault [f1, fk]. From (1) we obtain vp(f1) \neq vp(f2) and vp(f2) \neq vp(fk). Thus, ap(f1) = vp(f2) in (3.2.2) gives ap(f1) \neq vp(f1) and ap(f1) \neq vp(fk). For the fault [f1, fk], these relations exclude the condition b1) and b2).

Therefore, in all possible situations concerning the case \textbf{c.1 and c.2}, there is at least a double fault \textit{([f1, f2], [f2, fk])}, or \textit{([f1, fk])}, which does not satisfy neither condition a) nor condition b) of proposition 7. Thus, proposition 8 is proven for the case \textbf{c.1 and c.2}.

\[ \text{[f1, f2, ... fk] satisfies c.1:} \]

Let us consider the double faults [f1, f2], [f1, fk], and [f2, fk]. Let a1, a2, and ak be respectively the aggressor-sensitizing states of f1, f2, and fk. Since a1, a2, and ak take binary values (0 or 1), two of them will necessarily have equal values. Let, for instance a1 = ak. Furthermore, since condition \textbf{c.2} is satisfied, the aggressors of f1 and fk have the same bit positions. Then, from the proof of proposition 7, f1 and fk will be detected by a single read. This implies that the single-read double-fault detection property is satisfied when condition \textbf{c.2} is true.

\[ \text{[f1, f2, ... fk] satisfies c.1:} \]

Let us consider the double faults [f1, f2], [f1, fk], and [f2, fk]. Let a1, a2, ak be respectively the aggressor-sensitizing states of f1, f2, fk and v1, v2, vk be their victim-sensitizing states. Since a1, a2, ak, v1, v2, and vk take binary values (0 or 1), then, all possible cases concerning the variables a1, v1, a2 and v2 are:

- a1 = v1 and a2=v2. As in case \textbf{c.1} the aggressor cell of each fault has the same bit position as the victim cell of the fault, then, from the proof of proposition 7 f1 and f2 will be detected by a single read.
- a1 = v1 and a2=v2. In this case again, from the proof of proposition 7, f1 and f2 will be detected by a single read.
- a1 = v1 and a2=v2. Then we can have either ak = vk or ak=vk. From the proof of proposition 7, in the first case f1 and fk will be detected by a single read, and in the second case f2 and fk will be detected by a single read.
- a1 = v1 and a2=v2. This case is treated similarly to the previous one.

Therefore in all possible cases of condition \textbf{c.1}, f1 and f2, or f1 and fk, or f2 and fk are detected by a single read. Thus, from lemma 2 the single-read double-fault detection property is satisfied when condition \textbf{c.1} is true. QED

The outcome of lemma 16 is important as it shows a 100% coverage for faults of multiplicity higher than 2. Its impact is even wider, as it enables 100% coverage for faults of any multiplicity by means of a simple and low-cost approach proposed next.

\[ \text{D. Treatment of SRDF deceptive faults} \]

The proposed march tests achieve the single-read double-fault detection property for all faults of multiplicity higher than two, as well as for the vast majority of faults of multiplicity 2. We have also shown (Lemma 15), that these faults that cannot be covered by any march tests. (march-SRDF-deceptive faults). The non-covered faults consist in double faults \textit{[f1, f2]} having the following characteristics:

- f1 and f2 are of the type CFtr or CFwd (giving 4 double fault combinations), or f1 and f2 are of the type CFir (giving 1 double fault combination). As there are 15 single-cell and two-cells FFMs, which give 225 possible double faults combinations for f1, f2 (15 options for f1 x 15 options for f2), the non-covered double faults represent 2.2% of them (5/225).

- The bit positions of the two aggressor cells coincide, or the bit positions of the two aggressor cells coincide with the bit positions of the two victim cells. For a memory using m-bits words, there are m^4 distributions of the bit positions
of the aggressor and victim cells. The distributions in which the bit positions of the aggressor cells coincide are m². The distributions in which the bit positions of the aggressor cells coincide with the bit positions of the victim cells are 2m². Thus, the proportion of bit-positions distributions leading to non-covered faults is (m² + 2m²)/m² = (m²+2)/m²
- The sensitizing quadruplets leading to non-covered faults correspond to the half of the total number of sensitizing quadruplets.

From this analysis, the percentage of double faults that are not covered by the march test algorithms proposed in this paper (and by any march test - march-SRDF-deceptive faults) is equal to (5/225)(m²+2)/m²x0.5x100% = 10(m²+2)/9(m²)%.

For memory word size of 22-bits (16 data bits and 6 SECDSED check bits), and 39-bits (32 data bits and 7 SECDSED check bits), this percentage is respectively equal to 0.055%, and 0.03%. Thus, they represent a very small fraction of a particular case of faults (double-faults affecting the same memory word), and may be ignored in many applications. However, if the application requires covering them, we can employ one of the following approaches.

A first approach consists in developing test algorithms dedicated to this small fraction of faults. To cope with this issue we can use non-march algorithms that have the same simple structure as march algorithms (to preserve simple coincidences), which is not possible with march tests. To cope with this issue we can use non-march algorithms that have the same simple structure as march algorithms (to preserve simple BIST implementation), but instead of using inside each march element the same vector Vi for all memory words, we can use shifted versions of Vi. Due to obvious reasons of space limitations, the development of such algorithms is not considered in the present paper.

A second approach exploits the fact that, as shown in proposition 8, all faults affecting the same memory word and having multiplicity higher than 2, are covered by the proposed algorithms. Thus, as all non-covered faults produce errors that are detectable by ECC, we can employ the following strategy: These faults are not detected during the test and repair phase and are left un repaired. Then, each time a double error is detected by the ECC during system operation, the affected memory word is replaced by an error. This strategy is challenged by two potential problems:
- Repairing the memory words producing double errors during system operation will imply repairing not only those comprising double faults but also those containing a single faulty cell and are affected by an SEU. This will result in wasting spare resources. However, from the analysis in section II, in a very large memory of 100 Gbit and for a very high defect density of 10⁻³, this will happen for less than 2 times per three years. Thus, in a ten years life of a 100 Gbit SRAM, the wasted spare words will be less than 7. Adding 7 extra word in the CAM used for repair, represents an insignificant percentage of area and power cost with respect to a 100 Gbit SRAM.
- The existence of non-repaired memory words containing two faulty cells (march-SRDF-deceptive faults), may lead in undetectable triple faults if a soft error affects such a word. However, the probability of this kind of event is extremely low. Indeed, such an event will occur if an SEU affects at runtime an un repaired memory word W during an opportunity-time-window [t₁ t₂] such that:
  i. At t₂ word W is read.
  ii. At t₁ a write operation is performed over W and no other write is performed over W between t₁ and t₂.
  iii. W is sensitized during the time-window [t₁, t₂].
  iv. [t₁, t₂] is the very first time-window satisfying conditions i., ii., and iii., and happening after the occurrence of the double fault (which can occur during manufacturing or in the field). This is because, if no SEU occurs on W during this very first opportunity-time-window, then, as the double fault is sensitized during [t₁, t₂], the read operation at t₂ will detect a double error and the faulty word W will be repaired.

The duration of [t₁, t₂] determines the occurrence probability of the triple error. This duration depends on the application program, and can be evaluated through application simulations (e.g. of benchmark software applications), by taking into account the sensitizing conditions of the march-SRDF-deceptive faults. However this kind of evaluation is not in the scope of this paper. Instead we will consider a very pessimistic value for the mean duration of [t₁, t₂], such as one week! Considering this pessimistic value for [t₁, t₂], and a very large memory of 100 Gbit capacity, using words of 32 data bits and 7 SECDSED check bits, and affected by a high defect density 10⁻³, we find that the number of triple undetectable errors that will occur during the full product life is equal to 0.9999⁸⁻¹⁰⁻³ x₁₀⁻³⁻¹⁻⁹⁻²⁻²⁻₂⁻²⁻²⁻²⁻⁰⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻˓ of the SRAM cell and the flag cell area be q times larger than the SRAM cell and the flag cell area be q times larger than the
SRAM cell (usually $r$ is about 2 or slightly larger and $q$ is 1.5 or less). Then, the probability that the tag field does not contain faulty cells is equal to $(1 - P \delta)^{N_{\text{wc}}}$, where $N_{\text{wc}}$ is the number of cells of the tag field, and the probability that the flag cells are fault-free is equal to $(1 - P \delta)^{N_{\text{N}}}$. Let $N_{\text{wp}}$ be the number of words of the memory and $N_{\text{wc}}$ be the number of words of the repair CAM. Then, considering uniform fault distribution we find easily that the sum

$$
\sum_{r=t}^{N_{\text{wc}}} \frac{N_{\text{wc}}!}{r!} (1 - P_{\text{wp}})^r (1 - P_{\text{wg}}) \div (N_{\text{wc}} - r)! (1 - P_{\text{wc}})^r \div (N_{\text{wc}} - r)! r!
$$

there are at least $t$ good CAM locations. Then, the expression

$$
\frac{N_{\text{wc}}!}{r!} (1 - P_{\text{wp}})^r (1 - P_{\text{wg}}) \div (N_{\text{wc}} - r)! (1 - P_{\text{wc}})^r \div (N_{\text{wc}} - r)! r!
$$

gives the probability that there are $t$ memory words requiring repair and at least $t$ good CAM locations (i.e. the probability that the memory contains $t$ faulty words and is repaired). Then, as the number of faulty memory words that can be repaired cannot exceed the number $N_{\text{wc}}$ of the CAM words, the memory yield after repair is obtained by summing this expression from $t=0$ to $N_{\text{wc}}$, resulting in the expression:

$$
Y = \sum_{r=0}^{N_{\text{wc}}} \frac{N_{\text{wc}}!}{r!} (1 - P_{\text{wp}})^r (1 - P_{\text{wg}}) \div (N_{\text{wc}} - r)! (1 - P_{\text{wc}})^r \div (N_{\text{wc}} - r)! r!
$$

Expression 1 is valid for both conventional repair and ECC-based repair, provided that the probabilities of good memory word ($P_{\text{wc}}$) and good CAM word ($P_{\text{wc}}$) are computed as discussed earlier for the conventional repair and for the ECC-based repair.

To evaluate the computation complexity of expression (1) we need to determine the number of operations required for its computation. To determine this number we consider that, the factorials in the nominator the denominator in the two fractions in expression (1) are simplified by eliminating the one factorial in the denominator as well as all the terms of this factorial from the nominator of the factorial of the nominator. The optimal manner to simplify these factorials is to eliminate from the denominator the factorial ($N_{\text{wc}} - t)!$ if ($N_{\text{wc}} - t) \geq t$, or the factorial $t)!$ otherwise, and do the similar for ($N_{\text{wc}} - r)! and $t)!$. Also, let $k$ be the number of terms (without considering the term 1) of the factorial left in the denominator. In the computations we can either perform $k-1$ multiplications to compute the product of these terms and then divide the value of the nominator by the value of this product (requiring $k-1$ multiplications and 1 division); or divide successively the value of the nominator by each of these terms (requiring $k$ divisions). Next we employ the former manner as it minimizes the number of divisions.

**Derivation of the number of multiplications:**

The number of multiplications required for the term

$$
C_t = \frac{N_{\text{wc}}!}{r!} (1 - P_{\text{wc}})^r \div (N_{\text{wc}} - t)! (1 - P_{\text{wp}})^r (1 - P_{\text{wg}}) \div (N_{\text{wc}} - r)! (1 - P_{\text{wc}})^r \div (N_{\text{wc}} - r)! r! + (\text{Nw-1}) + (\text{t-2}) + 1 = (N_{\text{wc}} - 3) + 2t
$$

(Nw-3) +2t for all values of $t$ except for $t=0$ for which we should add 2, and for $t=1$ for which we should add 1. Thus, summing them for $t=0$ to $t=N_{\text{wc}}$ gives ($N_{\text{wc}} + 1)(N_{\text{wc}} + 1)(N_{\text{wc}} + 1)$ multiplications.

The number of multiplications required for the term

$$
B_t = \frac{N_{\text{wc}}!}{r!} (1 - P_{\text{wc}})^r \div (N_{\text{wc}} - r)! t!
$$

gives the probability that the CAM location is good for repairing a faulty memory word. In conventional repair, $P_{\text{wc}}$ is the probability that the CAM location is fault-free. Thus, we have $P_{\text{wc}} = (1 - P_{\text{wc}})N_{\text{wc}}$. In ECC-based repair, $P_{\text{wc}}$ is the probability that the tag field and the flag bits are fault free, and the data field is fault-free or it contains one faulty cell. Thus we have $P_{\text{wc}} = (1 - P_{\text{wc}})^{N_{\text{wc}}}((1 - P_{\text{wc}})^N + N(1 - P_{\text{wc}})^N - 1)$. The optimal one factorial in the denominator as well as all the terms of this computation.

**Derivation of the number of additions:**

Each term $C_t$ requires one division, giving $N_{\text{wc}} + 1$ divisions for all terms $C_t$ (i.e. for $t = 0$ to $t = N_{\text{wc}}$).

Each term $B_t$ requires 1 division. Thus, the sum

$$
\sum_{r=0}^{N_{\text{wc}}-1} B_r
$$

requires $N_{\text{wc}} + 1 - t$ divisions. Then, summing $N_{\text{wc}} + 1 - t$ for $t = 0$ to $t = N_{\text{wc}}$, to take into account the external sum, gives ($N_{\text{wc}} + 1)(N_{\text{wc}} + 1) - N_{\text{wc}}(N_{\text{wc}} + 1)/2$. Adding to this number $N_{\text{wc}} + 1$ (the divisions for all terms $C_t$), we find a total number of divisions equal to ($N_{\text{wc}} + 1)(N_{\text{wc}} + 4)/2$.

**Derivation of the number of additions:**
Nw - t additions are required for summing the Nwc - t + 1 terms Br of the internal sum \(\sum_{r=0}^{Nwc} B_r\) of expression 1. From the external sum \(\sum_{r=0}^{Nw} B_r\) of expression 1 there are Nwc + 1 internal sums (from t = 0 to t = Nwc). Thus, summing Nw - t + 1 from t = 0 to t = Nwc gives Nw(Nwc +1) - Nw(Nwc +1)/2 additions. Adding the Nwc additions for summing the Nw + 1 terms of the global sum \(\sum_{r=0}^{Nw} B_r\) gives a total of \(N_{WC}(N_{WC} +3)/2\) additions.

The outcome is that the computation of expression (1) requires: \(Nw(Nwc + 1) + (Nw^2 - 1)(5Nwc + 12)/6 - 1\) multiplications; \((Nw + 1)(Nwc + 3)/2\) divisions; and \(N_{WC}(N_{WC} + 3)/2\) additions.

This complexity is much higher with respect to the yield computation for low defect densities, where faults affecting the CAM have insignificant impact to the yield and are ignored, resulting in much simpler yield computation expression:

\[
Y = \sum_{t=0}^{Nwc} \frac{N_w!P_{wg}^{N_w-t}}{(N_w-t)!t!}(1-P_{wg})^t
\]

Also, as we deal with future very advanced technologies allowing producing very complex chips, we should be able to deal with very large memories. Also, as we deal with high defect densities we should be able to deal with large repair CAMs. In this context, the above numbers of operations are too large. For instance, for conventional repair of a 10 Gbit memory employing 32-bits words and affected by a \(10^3\) fault density, we have \(Nw = 335544320\), and \(Nwc \approx 350000\). Thus, computing the yield by means of (1) requires \(3.58 \times 10^{16}\) multiplications, plus \(6.12 \times 10^{10}\) divisions and the similar number of additions. Furthermore, these operations have to manipulate very large numbers such as \(Nw!(Nwc-t)!!\), as well as very small numbers such as \(P_{wg}^{Nw}\), requiring high precision arithmetic. Thus, computing the yield by means of expression (1) becomes computationally intractable.

To accelerate the computation of expression (1) we discovered certain recursive relations described below, which, to the best of our knowledge are unknown in the literature.

Setting

\[
A_t = \frac{N_w!P_{wg}^{N_w-t}}{(N_w-t)!t!}(1-P_{wg})^t
\]

\[
B_r = \frac{N_{WC}!P_{wcg}^{N_{WC}-r}}{(N_{WC}-r)!r!}(1-P_{wcg})^r
\]

we can write expression (1) as:

\[
Y = \sum_{t=0}^{Nwc} A_t \sum_{r=0}^{Nw} B_r
\]

We find that \(A_t\) and \(B_r\) can be written recursively as:

\[
A_0 = P_{wg}^{N_w}, A_{t+1} = \frac{(N_w - t)(1-P_{wg})}{(t+1)P_{wg}} A_t
\]

\[
B_0 = P_{wcg}^{N_{WC}}, B_{r+1} = \frac{(N_{WC} - r)(1-P_{wcg})}{(r+1)P_{wcg}} B_r
\]

We also set \(B_r \overset{\circ}{=} \sum_{r=0}^{Nw} B_r\) and we find that \(B_1\) can be written recursively as:

\[
B_0 \overset{\circ}{=} \sum_{r=0}^{Nw} B_r, B_{r+1} = B_r \overset{\circ} - B_{Nwc-r}
\]

Based on these relations the computation is done in the following manner:

- The terms \(B_0, B_1, B_2, \ldots, B_{Nwc}\) are used intensively in the computations. Thus, to avoid recomputing them multiple times we compute them once for ever by means of relations (4), and we store them in a lookup table. The creation of this table requires \(4N_{WC}-1\) multiplications and \(N_{WC}\) divisions.
- For each value of \(t\) we use the relations (6) and the stored values of \(B_0, B_1, B_2, \ldots, B_{Nwc}\) to compute each term \(B'_1\): the term \(B'_0\) requires \(N_{WC}\) additions and each other term \(B'_1\) one subtraction, resulting in a total of \(N_{WC}\) additions and \(N_{WC}\) subtractions for computing all the terms \(B'_1\).
- For each value of \(t\) we use the relations (3) to compute each term \(A_t\). \(A_0\) requires \(N_w -1\) multiplications and each other term \(A_t\) three multiplications and one division, resulting in a total of \(N_w - 1 + 3N_{WC}\) multiplications and \(N_{WC}\) divisions for computing all the terms \(A_t\).
- We use relations (2), (5) and the values of \(A_t\) and \(B'_1\) to compute the yield. This will require \(N_w + 1\) multiplications and \(N_{WC}\) additions.

Note that the values of \(A_t\) and \(B'_1\) do not need to be computed in advance and stored in look-up tables as they are used only once and we employing them in the good order, i.e. from \(t = 0\) to \(t = Nwc\), as required in the recursive relations (3) and (6).

Area and Power Evaluations: Thanks to the derived recursive relations and the use of look-up tables, the computation of the yield can be done in linear complexity with respect to the memory size: \(Nw + 8N_{WC} - 1\) multiplications, \(2N_{WC}\) divisions, \(2N_{WC}\) additions, and \(N_{WC}\) subtractions. This is dramatically shorter with respect to the number of operations required for computing expression (1) in direct manner. For instance for the above example of the 10 Gbit memory employing 32-bits words and affected by a \(10^3\) fault density, we can compute the yield by means of \(3.36 \times 10^8\) multiplications, \(7 \times 10^5\) divisions, \(7 \times 10^5\) additions, \(3.5 \times 10^5\) subtractions, resulting in a reduction of \(8\) orders of magnitude. This reduction allows computing the yield in short time even for very large memories and high fault densities.

The recursive yield computation approach was implemented in C++. We used this tool to determine the size of the repair CAM (for conventional and for ECC-based repair), for two SoCs having each a total embedded memory.
capacity of 9.75 Gbit (i.e. a total of 250M words x 39 bits per word, corresponding to 32 data bits and 7 Hamming code bits). In the one SoC this memory capacity is distributed over 300 memories, and in the other SoC its is distributed over over 3000 embedded memories (as reported in column 2 of table 1). The size of the repair CAM was computed, for four fault densities (10^-4, 3x10^-4, 10^-3, and 3x10^-3), and for 97% yield. These evaluations were performed for conventional repair, as well as for the ECC-based repair approach employing the proposed SRDF algorithms. Then, we used CACTI to estimate the area and power overhead for these implementations. The results are presented in table 5.

In this table we observe that, ECC-based repair employing the SRDF algorithms achieves drastic reduction of the area overhead with respect to conventional repair. The reduction is even higher for the power overhead, which is the most critical parameter in modern technologies.

### Table 5. Area and power cost comparison

<table>
<thead>
<tr>
<th>Pf</th>
<th>Embedded Mem</th>
<th>Conventional Repair</th>
<th>ECC-based Repair</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Ncw</td>
<td>%A</td>
<td>%P</td>
</tr>
<tr>
<td>10^-4</td>
<td>300</td>
<td>3466</td>
<td>1.32</td>
</tr>
<tr>
<td></td>
<td>3000</td>
<td>402</td>
<td>1.27</td>
</tr>
<tr>
<td>3x10^-4</td>
<td>300</td>
<td>10283</td>
<td>3.93</td>
</tr>
<tr>
<td></td>
<td>3000</td>
<td>1121</td>
<td>3.46</td>
</tr>
<tr>
<td>10^-3</td>
<td>300</td>
<td>35325</td>
<td>12.72</td>
</tr>
<tr>
<td></td>
<td>3000</td>
<td>3693</td>
<td>13.49</td>
</tr>
</tbody>
</table>

### B. Test Duration

For 39-bits memory words (32 data bits and 7 ECC check bits), the size of a 2-covering set is 14, and the size of the 3-covering and 4-covering sets proposed in [29][30] are respectively 25 and 66.

Using these numbers, we find that the length of the SRDF march tests proposed in section 4 is equal to 873 Nw, where Nw is the number of memory words. On the other hand, the conventional repair uses conventional (i.e. non-SRDF) test algorithms. The test length of the optimal conventional test algorithm for the FFMs of tables 1 and 2 proposed in [28], is 46 times shorter (19Nw). However, as the power dissipation during the test and repair sessions is larger for this approach, its test time is increased by a factor of 3.71, which is absolutely preferable than the huge 1629% power increase induced by the conventional repair approach. Also, the 12.75% area penalty (see table 5), induced by the conventional approach is totally undesirable. Indeed, as memories occupy the largest part of modern SoCs (more than 90% of the SoC area in most cases), the 12.75% area penalty induced by the conventional repair approach represents more than 11.5% of the total SoC area.

### VI. Conclusion

In this paper we show that, ECC-based repair may lose its advantages due to diagnosis issues. To cope with this issue, we introduced a new family of memory test algorithms (referred as SRDF algorithms), which eliminate the diagnosis hardware. Then, we addressed successfully the highly complex theoretical challenge concerning the analysis and derivation of SRDF test algorithms. In particular, we derived march SRDF tests for all static unlinked single-cell FFMs and two-cell FFMs. These algorithms achieve the SRDF property for all faults except for a very small fraction of double faults. Then, we proposed a technique achieving complete protection of these faults, at almost no cost.

The outcome is the dramatic reduction of area and/or power penalties at non-negligible increase of test duration, which is however fully justified by these reductions.

In a companion work we developed an iterative diagnosis approach that reduces test time at the expense of hardware cost, which is less efficient for high defect densities, but can offer interesting tradeoffs for mean defect densities. In another companion work we propose a new word repair architecture that reduces drastically runtime power for the repair CAM. All these developments lead to a comprehensive framework enabling memory repair for high defect densities at very low area and power penalties, which allows coping with the increasing unreliability and stringent low-power constraints of upcoming ultimate CMOS and post CMOS technologies.

### References
