An ECC-Based Repair Approach with an Offset-Repair CAM for Mitigating the MBUs Affecting Repair CAM

Panagiota Papavramidou, Michael Nicolaidis, Patrick Girard

An ECC-Based Repair Approach with an Offset-Repair CAM for Mitigating the MBUs Affecting Repair CAM

Panagiota Papavramidou[1] Michael Nicolaidis[1] Patrick Girard[2]

[1] *TIMA CNRS, Grenoble INP, UJF Grenoble*, France [2]*LIRMM, CNRS, Univ. of Montpellier*, France

Abstract— Memory system reliability is a serious concern in many systems today and is becoming more worrisome as technology scales, system size grows and the demand of aggressive voltage reduction becomes more stringent. Thus, disposing of memory repair architectures with strong fault tolerance capability at low cost is desirable. In this context, Error Correcting Codes (ECC)-based repair techniques were proposed and offer aggressive reduction of the repair cost for high defect densities. However, an important issue in advanced process nodes is the fact that, single particles induce Single-Event Upsets (SEUs) in neighbor memory cells, thus leading to Multi-Cell Upsets (MCUs) and Multi-Bit Upsets (MBUs), when they occur in the same memory word. In the case of memories, there exist efficient approaches mitigating this kind of MBUs, in particular the use of interleaving. But when a memory is repaired, the impact of MBUs on the circuitry repairing the faulty memory words should also be mitigated. This can be done by using a repair Content Addressable Memory (CAM) having interleaving at its data-words, or else an Offset CAM. In this paper we present and evaluate a novel repair approach that uses the Offset CAM in ECC-based Memory Repair and hence permits the mitigation of the MBUs affecting it.

Keywords—Memory repair; MBUs; Mitigation; Offset CAM; ECC;

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 manufacturing faults, affecting yield adversely. Hence, memory Built-In Self-Repair [1-6] is mandatory for maintaining acceptable fabrication yield. Moreover, field failures (soft-errors caused by neutrons and alpha particles, weak-cell faults activated during very-low voltage modes, circuit aging and wear-out) are a critical concern in memories. Thus, ECC becomes also compulsory for maintaining acceptable reliability. 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 manage faults affecting a single cell of a memory word and a CAM can be used to repair only memory words comprising two or more faulty cells. This is the ECC-based repair technique introduced in [7] and it has been shown in [8] that even for extremely high defect densities, it achieves high yield at much lower than the conventional repair area and power cost. Furthermore, in advanced deep submicron technologies, single-particle strikes induce often in memories Multi-cell Upsets (MCUs) [9-11] and thus multiple errors at the same memory word (usually double errors) known as Multi-Bit Upsets (MBUs). One way for mitigating MBUs is to use an ECC that is able to correct multiple errors, but such codes induce quite higher hardware and speed penalty than the Single-Error Correcting and Double-Error Detecting (SECDED) codes that are usually used. Therefore, to avoid these extra costs, the mitigation of MBUs in memories is commonly performed by using interleaving [12],[13], which consists in a memory architecture that has at each row more than one word and two consecutive bits that do not belong to the same word. Moreover, the systems which use CAMs for repairing memory words employ the same ECC for the words stored in the memory and in the CAM, but they do not use interleaving in repair CAMs. However, the impact of MBUs on them should also be mitigated, since the scaling down of CMOS technology and the growth of memory's size has increased the susceptibility of cells to particle radiation. Hence, to maintain a high level reliability, the use of interleaving is required as well as in the data-words of the repair CAM.

The only existing CAM architecture with interleaving at its data-words is the Offset CAM [14]. This is a CAM with several data-words in a row. During each memory read/write operation, an associative search is performed by means of the row address used at the current memory read/write. Each time a Hit occurs at this associative search, the selection of a data-word at the CAM row that is selected by this Hit is done by using as Offset bits the row address of the current memory operation.

So far, there is no approach using Offset-CAM for repair. To this end, we present a detailed scheme of memory repair by means of an Offset-CAM in order to mitigate the MBUs that affect it. In this repair, for each memory row that comprises faulty word(s), its row address is stored in the tag field of an Offset Repair-CAM row and its faulty word(s) is stored in the corresponding data-word(s) of this Offset CAM row. The position of the faulty word(s) in the memory row should be the same with the position of the data-word(s) in the Offset Repair-CAM row. Evidently, for a correct repair, it is required to avoid using any row of the Offset CAM that has faulty tag field and avoid repairing a faulty memory word by a faulty data-word of the Offset CAM. Hence the specific Offset CAM row should be fault-free at all positions that the memory row has faulty word(s). To achieve this, a fault diagnosis on the memory must be firstly performed, during which the faulty words of every memory row are identified and stored in a block. Afterwards, this information is used in order to select for each of these memory rows, a CAM row that will be able to repair its faulty words. In order to realise this repair, the architecture is completed by adding some flag cells at the CAM rows and two counters for visiting the rows and data-words of the Offset CAM. These are presented in section II of the paper. If we add a circuit in which will be stored the information related to the memory fault diagnosis, a significant extra hardware cost will be induced. Thus, to avoid this, we developed an approach (presented in section III) that achieves storing this information at the Offset CAM and using it later to determine the rows that will be used to repair the faulty memory words. Note that, discovering an approach that will determine consistently which row of the Offset Repair-CAM should be selected for repairing any faulty memory row is not obvious. Such an approach is presented in section IV. Here we study the ECC-based repair case but the approach can be applied also in conventional CAM repair. Section V presents the development of mathematical expressions for the yield computation of the proposed

architecture followed by the evaluations in Section VI. The paper ends with conclusions in Section VII. Hereafter, the Offset Repair-CAM row will be mentioned as **R-ORC** and the Offset Repair-CAM as **ORC**.

II. PARTS AND FLAG CELLS OF THE OFFSET REPAIR CAM

Each **R-ORC** will comprise:

- A **Tag Field,** in which the address of the faulty memory row that will be repaired by this **R-ORC** will be stored;
- A number of **Data-Words** (equal to the number of words of each memory row), which will be used to write and read the data of the faulty words of the memory row repaired by this **R-ORC**;
- A flag cell (**flag2**) at each data-word of the **R-ORC**, which will be set initially to 0. This will indicate that its data-word is not yet selected for repairing a faulty memory word and it will be set to 1 when its data-word is selected for repairing a faulty memory word.

Also the following flag cells are added in each **R-ORC**:

- **Flag1 (and Flag1')**: A flag cell (**flag1**) is put at each data-word of the **R-ORC** and it will indicate if this data-word is faulty or not. All flag1 cells of the **ORC** are initialized to 1. This flag indicates that its related data-word is fault-free. Each time during the test-session of the **ORC** a data-word is detected to be faulty (i.e. one or more of its data cells or its flag2 cell is faulty) its flag1 cell is set to 0 thus indicating that this data-word is faulty. But, if a flag1 is faulty, it may indicate that a bad CAM data-word is good for performing repair and finally it will result in incorrect repair. This issue is resolved by replicating the flag1 and adding a second flag1'.
- **Flag.t:** If the tag field of an **R-ORC** is detected to be faulty, the flag cells flag1 and flag1' of each data-word of this **R-ORC** will be set to 0. This will guaranty during each memory operation that none of these data-words will be selected, even if the Hit signal of this **R-ORC** is activated and thus induces a certain level of safety. But for each **R-ORC** whose tag-field is faulty or all its data-words are faulty, no row address will be set at its tag field and the tag field of this **R-ORC** will comprise the all 0s' state. Thus, during a memory operation at which the row address is equal to the all 0s' state, the Hit signals of several of these **R-ORC** will be activated. Also, even if the row address is not equal to the all 0s' state, the Hit signals of some of the **R-ORC** that have faulty tag field can be activated due to the faults in their tag field. These situations are disturbing as the activation of multiple Hit signals will lead to an incorrect operation of the **ORC** and will affect accordingly the execution of the current memory operation. Thus, for avoiding these situations, the flag.t cell is added at the tag field of each **R-ORC**. It is set initially to 1 and each time the tag field of an **R-ORC** is detected to be faulty, its flag.t is set to 0. Furthermore, at the end of the test session of the **ORC**, if a row with all its data-words faulty is found, the flag.t of this row is also set to the value 0. When the value of flag.t is 0, the Hit signal of its tag-field is deactivated by means of an AND gate whose inputs are the Hit signal generated by the tag field and the content of its flag.t. Also if a designer prefers to ensure this kind of mitigation even if flag.t signal is faulty, a second flag.t' can be added.

FAC-Counter and Word-Counter: During the executions of applications, the operations of the CAMs are performed by means of an associative search that identifies if a tag field of the CAM comprises the memory address used by the current operation. On the other hand, before starting the execution of an application, certain memory addresses should be stored at the tag fields of the CAM. But in this case the selection of a CAM row for storing a memory address at its tag field cannot be done by means of an associative search. In [14], authors show that the selection of an **R-ORC** can be done by means of a counter, which is referred to as FAC-Counter and an address decoder, which is referred to as FAC Address-Decoder. The number of bits of this FAC-Counter and the number of inputs of this FAC Address-Decoder are equal to $\log_2(rc)$, where *rc* is the total number of the **R-ORC**. For visiting the **R-ORC**, the FAC-Counter will be initialized to the all-0s' state and then it will be incremented. Each time a row is visited, it may also be needed to visit its data-words. For this reason, we add a second counter (mentioned as Word-Counter), whose size is equal to the number of bits of the column address of the memory under repair. The Word-Counter is also initialized to the all-0s' state and then it is incremented. We also add a multiplexer on the inputs of the Column Decoder of the **ORC**. During any regular memory read/write operation, the **ORC** uses the associative search and the multiplexer connects the column address of this read/write operation to the inputs of the Column Decoder. Whereas, during the operations in which the R-**ORC** and their data-words are needed to be visited by means of the FAC-Counter and the Word-Counter, this multiplexer connects the outputs of the Word-Counter to the inputs of the Column Decoder.

III. FAULT DIAGNOSIS FOR THE MEMORY AND THE OFFSET CAM

A. Fault-Diagnosis for the Offset Repair-CAM

In order to avoid using faulty **ORC** locations to repair faulty memory words, before starting the memory test-and-diagnosis session, the test session of the **ORC** will be executed. As mentioned, each time a faulty part of the **R-ORC** is detected its related flag cell is set to 0. At the end of this test session, all the tag fields and data-words of the **ORC** are initialized to all-0's state.

B. Memory Fault-Diagnosis Requirements for Achieving Consistent Repair

As far as it concerns the repair of faulty memory words, the faulty words of each memory row should be repaired by the fault-free data-words of an **R-ORC**. Thus, for achieving a consistent repair, we should know the positions of all faulty memory words comprised at each row. But during the execution of the memory test algorithm, in many cases the faulty words of the same row will be detected in different sequences of the test algorithm. Consequently, when a faulty memory word will be detected, it will not be possible to know if this memory row contains only this one or also some other faulty words and at which positions. Hence, selecting the **R-ORC** that would be appropriate for repairing this faulty memory word (and others that may belong to the same row) will not be possible. So, to achieve a consistent selection of the **R-ORC** that will repair the faulty memory words, a complete diagnosis of the faulty words at each memory row will be required. We set up an approach that uses the **ORC** to perform the diagnosis and afterwards analyse the diagnosis data stored in it for determining which **R-ORC** should be selected for repairing the faulty words of each memory row.

C. Complete Diagnosis of the Faulty Words in each memory row with the help of the Offset Repair-CAM

During the execution of the memory test algorithm, all the data of the faulty words in each memory row are stored in the **ORC**, with the following approach:

A- When a faulty word is detected, the FAC-Counter visits each **R-ORC (**by being reset to the all-0s' state and incrementing by 1) and at each visited row, it is verified from

the value of its flag.t cell if its tag field is fault-free. If this is the case, incrementing the FAC-Counter is stopped and the Word-Counter visits each data-word of this row by also being reset and incrementing by 1. Each visited data-word is checked if it is fault-free, from the values of its flag1 and flag1'. If a visited data-word is fault-free, incrementing the Word-Counter is stopped. However, if no fault-free data-word is found, even when all data-words of the current **R-ORC** have been visited, the searches performed by means of the FAC-Counter and the Word-Counter are re-executed until finding a row having fault-free tag field and at least one fault-free data-word. When such a row is found, the operations of step A are stopped until another faulty memory word is detected by the test algorithm. The information concerning this faulty memory word is stored in the first fault-free data-word of the **R-ORC** (selected by the FAC-Counter and the Word-Counter).

B- When a faulty memory word is detected by the memory test algorithm, the row-address of this faulty memory word is stored in the tag field of the **R-ORC**, selected by the current value of the FAC-Counter. If the value of the memory Column Address (C@) is equal to k (where $0 \leq k \leq (rw-1)$ and **rw** is the number of words per row), the kth output of the Column Decoder is equal to 1 and its other outputs are equal to 0 (the Column Decoder of the memory has rw outputs). Also, the word selected by this value (C@ = k) of C@ is the kth word of the memory row selected by the current value of the row address. Therefore, the output of the memory Column Decoder gives the position of the currently detected faulty memory word in its row and thus, in the data-word of the **ORC** that is selected by the FAC-Counter and the Word-Counter, we write these values. As already mentioned, the positions of all faulty words belonging to the same memory row should be stored at the fault-free data-word of the same **R-ORC.** This goal is achieved in the following way. Each time a faulty memory word is detected, the selection of an **R-ORC** by the FAC-Counter is deactivated and the associative search at the **ORC** is activated. If the positions of some other faulty memory word(s) of the same memory row with the currently detected faulty word are already stored at some **R-ORC**, the associative search will activate the Hit signal for the corresponding **R-ORC** and the position of the currently detected faulty memory word should be stored at the first fault-free data-word of the **R-ORC** that is selected by this Hit signal. To perform this, the bits of this data-word of the **R-ORC** will be read and each one will be "ORed" with the corresponding bit of the outputs of the Column Decoder of the memory. The result of this operation will be written back at this data-word. If no Hit occurs, the selection of an **R-ORC** by the current content of the FAC-Counter will be activated again and the position of the current detected faulty memory word will be stored in the first fault-free data-word of this row. Afterwards, step A- will be executed and when a new faulty memory word will be detected by the test algorithm, step B- will follow and so on…

IV. Consistent Repair Establishment with the Offset-CAM

A. Constraints concerning the Consistent Selection of the Offset Repair-CAM Data-Words that will Repair the Faulty Memory Words

At the end of the above diagnosis process, for each memory row containing some faulty word(s), the row address of these words is stored in the tag field of an **R-ORC** and the position(s) of these faulty word(s) are stored in the first fault-free data-word of this **R-ORC**. Subsequently, each **R-ORC** that contains the positions of the faulty words of a memory row will be visited and these positions will be used to find an **R-ORC** that could be selected for repairing the faulty words of the memory row. Let us note that:

a) An **R-ORC** may be able to repair the faulty words comprised at some different memory rows;

b) A memory row comprising certain faulty words may be reparable by several rows of the **ORC**. Then, if for a subset of memory rows that contain faulty words and certain rows of the **ORC** are selected for repairing them, it is possible that certain other memory rows comprising faulty words may be repaired only by the rows belonging to this selected subset of **R-ORC**. In this case, the repair will fail. We can make so that the repair succeed, if for some of the already repaired memory rows, we select some other **R-ORC** that would be able to repair them and liberate the **R-ORC** that are able to repair the non-repaired memory rows. However, in certain cases, achieving a successful repair can be quite complicate and may require several exchanges of the selected **R-ORC** that repair various memory words. Thus, for a consistent selection of the **R-ORC** that will lead to a successful repair, we developed several processes. The initial one is simple and achieves the repair of the majority of memory rows comprising faulty words. Then, we developed certain other processes that perform exchanges of the selected **R-ORC** and establish the repair of the minority of faulty memory words (that the initial process did not achieve).

B. Processes for achieving the Repair Accomplishment

FIRST PROCESS: The FAC-Counter visits the **R-ORC** until it finds the first row with the following four properties:

i) it has a **fault-free tag field** (flag.t=1 and flag.t' = 1)

ii) it has **fault free data-word(s)** (flag1 =1 and flag1' = 1)

iii) it **contains the positions of the faulty words** of a memory row. (The first fault-free data-word that is found is checked if there are one or more cells having value equal to 1. If such a cell is found, it is concluded that this row contains positions of faulty words of a memory row and the 1s' indicate the positions of the faulty words).

iv) it is **not yet selected for repairing a memory row** (verified by checking that all of the flag2 cells of its data-words are equal to 0).

This row will be mentioned as **Row.1**. First of all, it is checked if **Row.1** is able to repair the faulty words of the memory row whose positions are contained in this **Row.1.** To ensure this property we have to verify if all the data-words of **Row.1** that have the same positions (i) with the faulty words of the memory row are fault-free. In order to achieve this, we use the following three circuitries and they are the same that we will use when we will have to verify if another row (Row.X) is able to repair the faulty words of the memory row whose positions are contained in **Row.1**. This is because we want to avoid any additional hardware. The procedure is depicted in Figure 1, by replacing the Row.X by Row.1 or Row.2 according to the row that we want to check if it can repair Row.1.

a) The first circuitry includes rw AND gates for performing the function **dw.1(i)=flag1(i)**AND**flag1'(i)** of the pair of flag cells flag1, flag1' of each data-word i of **Row.1**. The output of these AND identifies the fault-free data-words of **Row.1**.

b) The second circuitry includes rw AND gates that perform the operations **dw(i)=dw.1(i)**AND**db.1(i)**, where dw.1(i) is the output of one of the above AND gates and db.1(i) is the value of the bit b(i) of the first fault-free data-word of **Row.1**.

c) Finally, the above circuits are completed by a **comparator** that compares the value **dw(i)** (output of circuit b) against **dw.1(i)** for all positions (i) that are used. Thus, when the output of this comparator is equal to 1, it can be shown that **Row.1** can repair the faulty words of the memory row whose positions are contained in this and it is selected to do this. If this is not the case, the following procedure is performed**.**

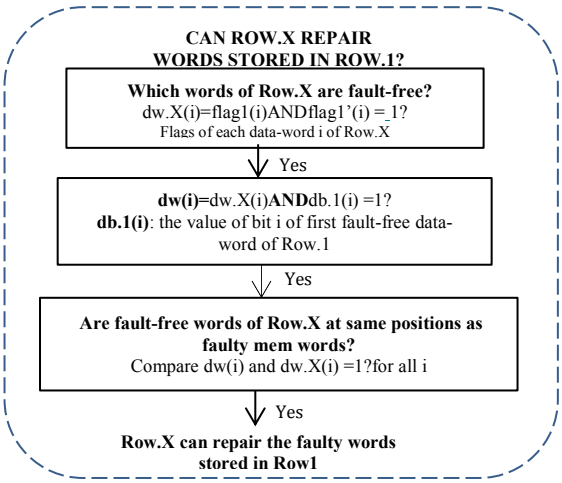

Figure 1: Procedure for verifying if a Row.X can repair the faulty memory words stored in Row.1

Before all, as in some cases of the repair process it is required to store the states of an **R-ORC** that comprises the positions of the faulty words of a memory row in a register (say Row.X), we add two registers in our circuit (**Registers R1** and **R2** that will be mentioned hereafter as **R1** and **R2**) having each of them four parts (**part-F**, **part-RA, part-DWB**, **part-DW**), as shown in Figure 2 (X stands for 1 or 2). **Part-F** is for storing the current content of the FAC-Counter (that selects **Row.X); part-RA** stores the current content of the tag field of **Row.X** (equal to the address of the memory row whose positions of faulty words are stored at **Row.X**); **part-DWB** stores the current content of the first fault-free data-word of **Row.X**; and at each bit i of the **part-DW** is stored the result of the operation **dw.X(i)= flag1(i)**AND**flag1'(i),** for each word i of this Row.X. So, each time an **R-ORC** (**Row.1)** is not found able to repair the faulty words of a memory row whose positions are contained in this, the related states of **Row.1** are stored at the parts of **R1**.

F	RA	DWB	DW
Content of FAC-Counter when shows this Row.X	Content of Tag-Field of Row.X	Content of first fault-free dw of Row.X	dw.X(i)=flag1(i)**AND**flag1'(i) for each word i of the Row.X
$\log_2$rc rc: # of R-ORC	#bits of Row Address	#bits of each data-word	#of words per row

Figure 2: The fields of Register RX

Then, the next **R-ORC** (**Row.2**) is searched, which should be a row that: has **not yet been selected for repairing** a memory row and either comprises the positions of the faulty words of a memory row or it does not comprise the positions of the faulty words of any memory row. If this is valid, the states related to **Row.2** are stored at the parts of **R2**. After that, **Row.2** is checked if it is able to repair the faulty words of a memory row whose positions are stored in **Row.1** and **R1**. In order to verify this, we use the circuitries **a, b and c)** presented above and the procedure is the one that is shown in figure 1. If this is the case, it means that **Row.2** can repair all the faulty words of the memory row stored in the **Row.1** and **R1** and it is selected for making so. This selection is achieved by: i) putting the content of the tag field of **Row.1** at the tag field of **Row.2.** ii) setting flag2=1 for each data-word of **Row.2. FIRST PROCESS** is depicted in figure 3.

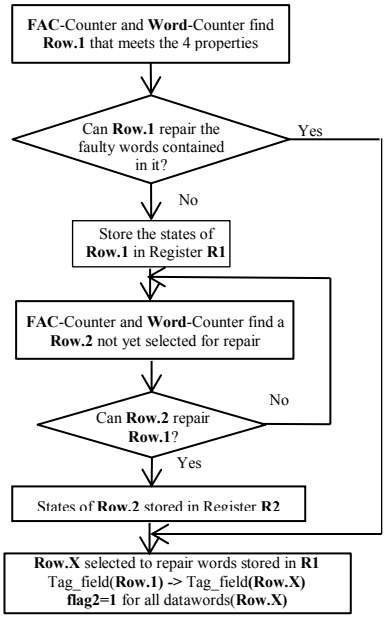

Figure 3: **FIRST PROCESS** for consistent selection of Row.X to repair the faulty memory words stored in Row.1

In the case where **Row.2** comprises already the positions of the faulty words of another memory row, it is checked similarly if **Row.1** is able to repair the faulty words of the memory row whose positions are stored at **Row.2.** If this is confirmed, the states **of Row.2** that are stored at **R2** are put also at **Row.1.**

SECOND PROCESS: So far, during the **FIRST PROCESS**, the rows of the **ORC** that were able to repair any faulty memory row have been identified and they have been selected for this repair. Therefore, if at the end of the **FIRST PROCESS,** there exist some faulty memory rows that are not repaired, it means that it is impossible to repair them by means of an **R-ORC** that is not yet selected. So, the only approaches that could find a solution to repair any non-yet-repaired faulty memory row are those that make some exchanges at the rows of the **ORC** that have been selected for repairing some faulty memory rows.

A first step before making such exchanges consists in finding each **R-ORC** that meets the four (**i-iv**) properties of the FIRST PROCESS. Each time such a row is found, it is mentioned as **Row.1** and its related states are stored at **R1**. Then, for repairing successfully the faulty words of the memory row whose positions are stored at **Row.1,** an **R-ORC** is searched (**Row.2**) that has been selected for repairing a row and:

a) **Row.2** is able to repair the faulty-words of a memory row whose positions were stored at **Row.1** (and **R1);**

b) **Row.1** is able to repair the faulty words of a memory row which are repaired by **Row.2** and whose states are stored at **R2**.

These properties are verified by the circuitries **a), b)** and **c)** of the **FIRST PROCESS** and the procedure shown in Fig.1. If they are both valid, then **Row.1** is selected to repair the faulty words of a memory row that were repaired by **Row.2**; and **Row.2** is selected to repair the faulty words of a memory row whose positions were stored in **Row.1**. In figure 4 is shown the **SECOND PROCESS**.

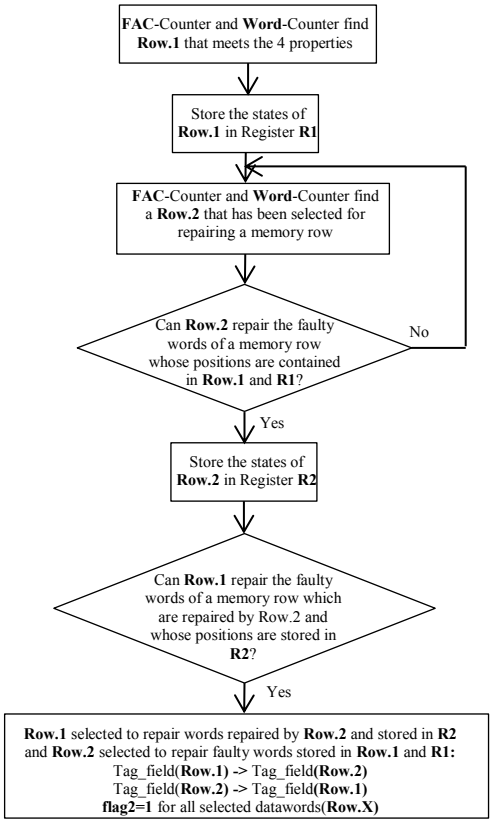

Figure 4: **SECOND PROCESS** for the repairing non-yet-repaired faulty memory row

After the execution of the **FIRST** and **SECOND PPROCESSES,** the large majority of the faulty memory rows will be repaired. Therefore, the probability that some faulty memory rows remain unrepairable will be low. However, if for a **Row.1** is not found any row mentioned as **Row.2** for which the above properties are valid, the faulty memory row whose positions are stored at **Row.1** should be repaired by another approach, which performs the exchange between 3 or 4 rows of the **ORC**. However, these approaches are not described in this paper due to lack of space.

V. YIELD ESTIMATION

We performed some experiments to evaluate the proposed approach. To this purpose, we had to derive the corresponding yield computation expression for achieving a target yield (e.g. Y_{SOC} = 95%) and then find the area and power penalties of the proposed architecture. Let rm be the number of memory rows, rc the number of **ORC** rows and rw the number of words per memory row and also per **R-ORC**. In ECC-based repair a memory word is considered to be "good" if it has 0 or 1 faulty cell in the data bits and 0 faulty cell in the tag and flag bits. P_{Gmw} is the probability that a memory word is good, P_{Gcw} is the probability that an **ORC** data-word is good, P_{Gct} the probability that a tag field of the **ORC** is fault-free and P_{Gcf} the probability that the flag cells of an **R-ORC** are fault-free. Also *Nb* stands for the number of bits of the memory and of **ORC** data-word. Then: $P_{Gmw} = P_{Gcw} = (1-P_f)^{Nb} + N_b(1-P_f)^{Nb-1}P_f$

Where P_f is the probability of a memory cell to be faulty;

$$P_{Gct} = (1-P_f)^{rN@} \quad \text{and} \quad P_{Gcf} = (1-P_f)^{qNf}$$

N@ is the number of cells of the tag field of the **ORC**, *r* is the ratio (tag cell area)/(SRAM cell area), *Nf* the number of its flag cells and *q* the ratio (flag cell area)/(SRAM cell area). *Q* is the number of faulty memory rows. If *Q* is larger than the number *rc* of the **ORC** rows, the repair is impossible. Thus, in the computation of the repair probability, we have to take into account values of *Q* that do not exceed *rc*, i.e. $0 \le Q \le rc$. We take the partition of the number *Q* of faulty memory rows into groups such that each group comprises faulty rows with a given number of faulty words. In each faulty row the minimum number of faulty words is 1 and the maximum is *rw*. Thus, it can exist *rw* groups with k(1), k(2),…, k(rw) faulty rows where each row of the group of k(1) faulty rows has 1 faulty word, each row of group k(2) faulty rows has 2 faulty words,…, each row of group k(rw) faulty rows has rw faulty words. As Q is the total number of faulty memory rows, all sets of integers k(1),k(2), …,k(rw) are satisfying the condition k(1)+k(2)+….+k(rw)=Q and k(i) are in the domain $0 \le k(i) \le Q$ for every $i \in (1,2,..,rw\}$.

A. Computing the Probability for Repairing all the k(1), k(2),…, k(rw) faulty rows

As already mentioned, for a successful repair, an **R-ORC** with fault-free tag field must exist for each faulty row of each group consisting of k(i) faulty rows (with i faulty words each), and the i words of this **R-ORC** corresponding to the i faulty words of the faulty memory row have to be fault-free while the rest of the words of this CAM row can be faulty or fault-free. Then, the probability that a memory row has i faulty words (and the remaining rw-i words are fault-free) and can have any possible positions over the rw words of the row, is equal to:

a) $P_{Gmw}^{rw-i}(1-P_{Gmw})^i \frac{rw!}{(rw-i)!i!}$, where (rw!/(rw-i)!i!) is the number of all possible positions of i faulty words among rw words of the row (i.e. number of permutations of i elements over rw).

For an **R-ORC** to be able to repair the i faulty words of a memory row, it should:

b) Have fault-free tag field (which probability is equal to P_{Gct})
c) Have fault-free flag cells (which probability is equal to P_{Gcf})
d) Have fault-free data-words (with probability equal to P_{Gcw}^i)
e) Each of the remaining (rw-i) data-words of this **R-ORC** can be fault-free or faulty
f)The i fault-free words of this **R-ORC** have the same positions as the i faulty words of this memory row.

From points b), c) d), e) and f) the probability that an **R-ORC** can repair the i faulty words of a memory row is found to be equal to the product $P_{Gct}P_{Gcf}P_{Gcw}^i$. And so, the probability that a memory row has i faulty words and these are repaired by an **R-ORC** is given by the expression:

$$(P_{Gmw}^{rw-i}(1-P_{Gmw})^i \, rw!/(rw-i)!i!)P_{Gct}P_{Gcf}P_{Gcw}^i$$

Then, the probability that there are k(i) memory rows having each i faulty words and being repairable by the **ORC**, is:

$$\left[P_{Gmw}^{rw-i}(1-P_{Gmw})^i \frac{rw!}{(rw-i)!\,i!}\right]^{k(i)} \cdot \left[P_{Gct}P_{Gcf}P_{Gcw}^i\right]^{k(i)}$$

Furthermore, the probability that k(1), k(2),....k(rw) specified memory rows comprise respectively (1, 2,…rw) faulty words and can be repaired by the **ORC,** is equal to the product:

$$\prod_{i=1}^{rw}\left(\frac{rw!P_{Gmw}^{rw-i}(1-P_{Gmw})^iP_{Gct}P_{Gcf}P_{Gcw}^i}{(rw-i)!i!}\right)^{k(i)}$$

The above product should be multiplied by the probability $P_{Gmw}^{rw(rm-Q)}$ of the remained rm – (k(1)+…+ k(rw)) memory rows with rw words per row and 0 faulty words.

$$P_{Gmw}^{rw(rm-Q)}\prod_{i=1}^{rw}\left(\frac{rw!P_{Gmw}^{rw-i}(1-P_{Gmw})^iP_{Gct}P_{Gcf}P_{Gcw}^i}{(rw-i)!i!}\right)^{k(i)} \qquad \textbf{(1)}$$

In order to find the total repair probability for all possible distributions of the k(1),k(2),…k(rw) faulty memory rows over the rm memory rows, we need to determine the number of these distributions. In mathematics, there is an expression for the number of combinations of k elements over r elements (*k-out-of-n*), but it cannot be used here because the subsets of elements associated to two different distributions of k-out-of-n elements are different, while the subsets of two different distributions of the k(1),k(2),… faulty memory rows over the rm memory rows are not necessarily different. We have shown that, with Q=k(1)+k(2)+…+k(rw), the total number of all possible distributions of k(1), k(2),…., k(rw) faulty rows over the rm memory rows is given by the expression:

$$rm!/((rm - Q)!\ k(1)!k(2)!\ldots k(rw)! \quad \textbf{(2)}$$

Proof is long and is not included in this paper due to space limitations. By using the product symbol Π, the expression (2) can be written as: $\frac{rm!}{(rm-Q)!\prod_{i=1}^{rw} k(i)!}$ **(3)**

The above expression is used also for the number of all possible distributions of the rc **R-ORC** that can repair respectively the k(1),k(2),…,k(rw) faulty memory rows:

$$\frac{rc!}{(rc-Q)!\prod_{i=1}^{rw} k(i)!} \quad \textbf{(4)}$$

For a given value of Q, the memory repair probability will be equal to the sum of the repair probabilities for all possible sets k(1), k(2),.., k(rw) with sum equal to Q. For the total repair probability, we should take the sum of the probabilities for each value of Q with repair probability other than 0. Thus, the total repair probability is given by the following expression:

$$P = \sum_{Q=0}^{rc} \sum_{k(1)+..k(rw)=Q} \frac{rm!\,rc!\,(P_{Gmw}^{rw})^{(rm-Q)}}{(rm-Q)!\,(rc-Q)!} \cdot \prod_{i=1}^{rw}\left[\frac{1}{(k(i)!)^2}\left(\frac{rw!}{(rw-i)!i!}P_{Gmw}^{rw-i}(1-P_{Gmw})^i P_{Gct}P_{Gcf}P_{Gcw}^i\right)^{k(i)}\right] \quad \textbf{(5)}$$

VI. EVALUATIONS

In this section, we evaluate the efficiency of the proposed approach. We used the analytical computation expression developed in section V to determine the sizes of the **ORC** for the target yield (95%). Then we employed the CACTI tool to find the area and power overhead of our approach. We considered an SRAM with 131072 words of (32 data bits+7 bits ECC SECDED) and interleaving of 2 words per row and the same interleaving for the **ORC**. Table 1 depicts the area and power costs for the case of the conventional CAM-based repair, the case of ECC-based Repair with a regular CAM without interleaving and the proposed scheme of ECC-based Repair with the **ORC**. Column 1 gives the defect density (P_f) and columns 2 to 7 give the area and power cost of each case compared to the area and power of the SRAM.

TABLE 1. Area and power cost for Conventional CAM-based Repair, ECC-based repair and ECC-based repair with an ORC

	Conventional Repair with CAM		ECC based Repair with Ordinary CAM		ECC-based Repair with Offset CAM	
P_f	%A	%P	%A	%P	%A	%P
$3*10^{-4}$	6.4915%	383.47%	0.099%	5.164%	0.12%	6.799%
10^{-3}	22.3%	1278%	0.447%	31.37%	0.575%	39.97%
$3*10^{-3}$	70.044%	3790%	3.92%	243.28%	5.05%	302.76%

The area and power penalty of the ECC-based Repair with the **ORC** compared to the conventional CAM-based repair remains still significantly low. With respect to ECC-based repair with regular CAM it is increased 20%-30%. However the very low cost and the increase of reliability permit this increase of penalties. In addition to this we can assume that the protection of an ECC (SECDED) in combination with the 2 word interleaving is equivalent to an ECC double error correcting triple error detecting code (DECTED). According to [14], the area penalty increase of our approach is similar to that of an ECC DECTED, the power penalty is almost the half and the latency of the advanced ECC is larger. Thus, the proposed repair approach with **ORC** achieves similar mitigation of MBUs with less overhead than advanced error correcting codes.

VII. CONCLUSION

As scaling progresses, circuits become more sensitive to particle radiation and need more protection. Accordingly, in the case of CAM-based memory repair, techniques for mitigating the MBUs in repair CAMs should also be used and more specifically, in order to avoid, the expensive multiple error correcting codes, the interleaving should be applied. The interleaving in CAMs can be achieved by means of the Offset CAM however so far, no such repair approach exists and it requires performing some specific processes for the memory fault diagnosis and the selection of the **R-ORC** that will repair the faulty words. We proposed a novel efficient approach performing repair by means of an **ORC** and we resolved the complex issues related to this kind of repair. This strong multibit error protection in the case of ECC-based **ORC** can be used to improve memory manufacturability and yield since it permits aggressively scale cell sizes while maintaining robust operation at low area and power cost in comparison with conventional CAM-based memory repair.

REFERENCES

[1] Zorian Y., "Embedded Memory Test & Repair:Infrastructure IP for SOC Yield", *IEEE ITC*, 2002.

[2] Sawada K. et al, "Built-In self repair circuit for High Density ASMIC", *IEEE Custom Integrated Circuits Conference,* 1999.

[3] Benso A. et al,"A Family of Self-Repair SRAM Cores", *IEEE IOLTW*, 2000.

[4] Kim I. et al, "Built-In self repair for embedded high-density SRAM", *IEEE ITC*, 1998

[5] V. Schober, S. Paul, O. Picot, "Memory Built-In Self-Repair using redundant words", *IEEE ITC,* 2001.

[6] M. Nicolaidis, N. Achouri, S. Boutobza, "Optimal reconfiguration functions for column or data-bit built-in self-repair", *IEEE DATE*, 2003

[7] M. Nicolaidis, N. Achouri, L. Anghel, "A Diversified Memory Built In Self Repair Approach for Nanotechnologies",*IEEE VTS,* 2004.

[8] P. Papavramidou, "Memory repair for high fault rates*", IEEE ITC*, 2016.

[9] J.Maiz, S. Hareland, K. Zhang, and P. Armstrong, "Characterization of multi-bit soft error events in advanced SRAMs," *IEEE Int. Electron Devices Meeting IEDM'03 Digest*, Dec. 2003.

[10] A.M.Chugg, et al.,"A statistical technique to measure the proportion of MBU's in SEE testing," *IEEE Trans. Nucl. Sci.*, vol. 53, no. 6, pt. 1, pp. 3139–3144, Dec. 2006.

[11] D. Giot, et al "Heavy ion testing and 3-D simulations of multiple cell upset in 65 nm standard SRAMs," *IEEE Trans. Nucl. Sci.*, vol. 55, no. 4, pt. p. 2048–2054, Aug. 2008.

[12] S. Baeg, et al "SRAM interleaving distance selection with a soft error failure model", *IEEE Trans. Nucl. Sci.*, vol. 56, no. 4, pp. 2111-2118, Aug. 2009.

[13] P. Reviriego, et al "Protection of Memories Suffering MCUs Through the Selection of the Optimal Interleaving Distance", *IEEE Trans. Nucl. Sci.*, vol. 57, no. 4, Aug.. 2010.

[14] T. Kohonen, "Content-Addressable Memories", *Springer-Verlag, 1987.*

[15] Jangwoo Kim et al, "Multibit Error Tolerant Caches Using Two-dimensional Error Coding, *IEEE/ACM MICRO Jan. 2007*