Reliability-Enhanced ECC-Based Memory Architecture Using In-Field Self-Repair

Gian MAYUGA(✉), Nonmember, Yuta YAMATO(✉), Tomokazu YONEDA(✉), Yasuo SATO(✉), and Michiko INOUE(✉), Members

SUMMARY Embedded memory is extensively being used in SoCs, and is rapidly growing in size and density. It contributes to SoCs to have greater features, but at the expense of taking up the most area. Due to continuous scaling of nanoscale device technology, large area size memory introduces aging-induced faults and soft errors, which affects reliability. In-field test and repair, as well as ECC, can be used to maintain reliability, and recently, these methods are used together to form a combined approach, wherein uncorrectable words are repaired, while correctable words are left to the ECC. In this paper, we propose a novel in-field repair strategy that repairs uncorrectable words, and possibly correctable words, for an ECC-based memory architecture. It executes an adaptive reconfiguration method that ensures ‘fresh’ memory words are always used until spare words run out. Experimental results demonstrate that our strategy enhances reliability, and the area overhead contribution is small.

key words: memory repair, memory reliability, in-field test and repair, ECC, in-field repair strategy, remapping

1. Introduction

Embedded memories are used to integrate memory blocks alongside logic cores or logic blocks on the same die or same module. It enables achievement of high performance and power optimization. With the continuous scaling of device technology, higher performance and denser memory contributes to more sophisticated System-on-Chips (SoCs). However, as good as its contributions to having better features, the memory takes up more than half of the area in large-scale SoCs. A larger area size of memory is more susceptible to errors, which affects both manufacturing yield and field reliability [1]. As memory occupies large area in an SoC, memory yield directly affects the overall yield of the SoC. To enable memory yield enhancement, repair using comprehensive redundancy scheme has been used. In order to enhance yield further, a combined approach of error correction code (ECC) and redundancy scheme is introduced [2]. Also, errors caused by hardware faults are becoming a threat to field reliability. They are introduced in field by reliability-related failure mechanisms such as aging, and permanently produce errors unless they are repaired, which is crucial for safety-critical systems. In-field test and repair, as well as ECC, are required to maintain reliability. In-field test and repair are performed during system operation, and memory issues can be fixed even after the devices have been shipped to the end customer. For instance, unused spare elements can be used to perform in-field self-repair [1]. ECCs are typically integrated into memory architectures to make them immune to errors. Combined approaches of using both repair and ECC give a synergistic effect to improve both yield and reliability, and have been explored in several works [2]–[6]. In this work, we present how the combined approach can be adapted into an in-field repair strategy, in order to enhance reliability. Next, we discuss how the combined approach changes ways to treat memory errors.

Memory errors are classified as either hard or soft errors [3]. Hard errors are errors caused by permanent hardware faults. The causes are 1) single bit fault for a cell in a memory word, 2) faults for all cells in a memory row/column, or 3) faults for all cells in a memory array. Causes of soft errors are transient phenomena such as alpha rays or cosmic rays that change the data stored in memory cells. Memory errors can either be repaired (hard errors) or corrected (hard and soft errors). For embedded memories, physical reconfigurations will be costly. Functional-reconfiguration techniques can be used where memory access is logically reassigned to a spare instead of the faulty row/column [7], word [4] or cell [8]. Correction capability depends on the number of correctable bits by the ECC. Single-Error Correction (SEC) is the most common type. ECC is effective for correcting both hard and soft errors, but at the expense of extra bits added to the memory data to form a codeword, as well as encoder and decoder circuits to generate and extract the codeword. As an additional protection, the error-free data output can be fed back to the source memory word and can be re-written, in order to ensure the integrity of the memory data. When the error-free data is re-written periodically, this process is called scrubbing, and it can either be done via software or hardware controller [10].

Given that ECC handles memory errors in memory words in field, this allows re-classification of memory words and errors. Erroneous memory words are classified as either 1) correctable words, which can be corrected by ECC, and 2) uncorrectable words, which the ECC cannot handle. We can then refer to a memory error as either a 1) correctable
error, if it is in a correctable word, and 2) uncorrectable error, if it is in an uncorrectable word. Handling of correctable and uncorrectable words is important since it affects memory lifetime, as explored in a recent study of a large experimental scale of errors in DRAM memory used in Google servers [9]. The conclusions that stand out are as follows: 1) A correctable word is more likely to develop uncorrectable errors (in the same month, there is a 70–80% chance that an uncorrectable error will appear after a correctable error appears in the same Dual Inline Memory Module (DIMM)); 2) Error rates are unlikely to be dominated by soft errors, and hard errors contribute the most for memory failures, and 1.3% of machines across the entire fleet are affected by uncorrectable errors per year, in such a way that even if a DIMM is replaced with a new one at the first sign of an uncorrectable error, another uncorrectable error appears on a different DIMM. These observations on module-based memory devices indicate that it is important to mitigate the occurrence of correctable words as much as uncorrectable words.

The combined approach strategy enhances both yield and reliability. The work [2] enhances yield by allowing correctable words to be shipped since ECC will be able to handle them during system use. The work [5] improves reliability by performing in-field self-repair only to uncorrectable words, letting ECC handle correctable words, and preserving remaining spare elements for future uncorrectable words. However, in both works, since correctable words are used by the system, memory words are always susceptible to occurrences of additional errors, which leads to occurrence of uncorrectable words, and thus, leads to memory failure. This causes a reliability problem. The reliability can be enhanced if the system can prevent itself from experiencing the scenario where the ECC has to handle uncorrectable words. Also, together with uncorrectable words, if correctable words are also repaired, the system is mitigated from going into failure, even on the occurrence of an additional error.

To solve this reliability problem, we propose an in-field repair strategy, given an ECC-based memory architecture. The main contributions are as follows:

- We propose a novel in-field repair strategy that enhances reliability by repairing uncorrectable words first, then repairing correctable words when there are spare words left. To guarantee repair of uncorrectable words, the previous repair of correctable words is cancelled if necessary.
- We formulate a stochastic reliability model of the proposed strategy, where reliability improvements contributed by memory scrubbing and repair of correctable words are evaluated, as well as that of repair for uncorrectable words only. The evaluation shows that the proposed strategy provides higher reliability than the traditional combined approach strategy.
- We implement a remap controller that executes an adaptive reconfiguration method to realize the in-field repair strategy. The overhead of the remap controller is found to be small.

2. Proposed In-Field Repair Strategy for Reliability Enhancement

In this section, we describe in detail a memory architecture that utilizes memory protection using the proposed in-field repair strategy of handling memory error, alongside error correction. The section is divided into 3 parts: 1) ECC-based memory architecture, 2) proposed in-field repair strategy, and lastly, 3) remap controller implementation. In our proposed work, we use single error correction.

2.1 ECC-Based Memory Architecture

The ECC-based memory architecture is based on work [5]. It is composed of memory, ECC circuit blocks, memory built-in self-test (BIST), diagnosis and remap content addressable memory (CAM) and remap controller (Fig. 1). We assume that an effective hardware repair has already been applied at production test, and memory used by the system includes enough number of spare words. Also, our proposed strategy can be applied to cases wherein correctable words are included in the shipped memory devices. The proposed architecture operates in two modes, 1) user and 2) test, which are detailed as follows:

1. User Mode: The user has access to the ECC-based memory architecture. Given read operation for memory word, remap CAM is searched as shown in Fig. 2(a). Data output is accessed from the user memory if there is no hit, and from spare memory if there is a hit. The remap CAM, ECC encoder/decoder circuits and scrubbing controller are all operational. The parts are detailed as follows:

   a. Remap CAM: The remap CAM is active and a given original address is remapped to a spare address if the original address has been identified as

![Fig. 1 Proposed ECC-based memory architecture](image-url)
Table 1 Remap CAM cases

| Faulty Word Address Entry | Number of Faulty Bits | Remap CAM Condition |
|---------------------------|-----------------------|---------------------|
|                           | Previous Test Period  | Current Test Period | Enough | Full-1 | Full-2 |
| new                       | -                     | ≥2                  | 1) write (RC2F) | 2) decrement (RC2F) | fail |
| new                       | -                     | 1                   | 1) write (RC1F) | 2) increment (RC1F) | do nothing |
| old                       | ≥2                    | ≥2                  | do nothing | do nothing | do nothing |
| old                       | 1                     | 1                   | do nothing | do nothing | n/a |
| old                       | 1                     | ≥2                  | 1) increment (RC2F) | 2) copy (hit_address, RC2F) | n/a |
| old                       | ≥2                    | 1                   | 1) increment (RC2F) | 2) copy (hit_address, RC1F) | same as Full-1 |

Full-1: both correctable and uncorrectable word addresses
Full-2: uncorrectable word addresses only

2. Test Mode: Memory systems are physically organized into several blocks [11]. We assume that each block becomes idle periodically, and test can be applied when it is idle. We can apply periodic self-test and diagnosis separately on each memory block. The test mode operates as shown in Fig. 2(b). BIST is firstly applied to find faulty words, then the found faulty word addresses and faulty bit locations are placed in the diagnosis CAM, and then based on the proposed strategy, the remap CAM is reconfigured based on operations listed in Table 1. The parts are detailed as follows:

a. BIST: BIST identifies faulty memory words. A March-like test repeatedly performs write and read operations in memory words. For each period of self-test, faulty bits in memory word are identified.

b. Diagnosis CAM: A diagnosis CAM is adapted [5] to classify faulty words based on the number of faulty bits. It stores the address of the faulty word, and the faulty bit locations within the word, once a faulty bit is detected during test. The address of the word is used as a tag to access the word in the diagnosis CAM. During test, though the same bit may be identified to be faulty several times, or multiple bits in the same words are identified to be faulty at different test timings, they are well integrated in the diagnosis CAM and every faulty word can be classified based on the number of faulty bits in a word.

c. Remap Controller: After the diagnosis CAM has stored all the faulty word addresses of the memory, this information is sent to a remapping controller. The remapping controller decides how the faulty word addresses are stored on the remap CAM depending on the number of faulty bit locations in the faulty word. It also updates the contents of the remap CAM on the succeeding self-test and repair periods.

2.2 Proposed In-Field Repair Strategy

In the proposed strategy, given enough spares, both uncorrectable and correctable words are repaired using functional remapping after applying self-test and repair. It improves
reliability since the possibility of uncorrectable words during system use is reduced. However, the number of uncorrectable words may increase, and not all erroneous words may be remapped to fresh spare words. In such a case, we give priority to uncorrectable words to be remapped, to the extent of cancelling previously remapped correctable words. The correctable words are then left to the system, wherein it is protected by error correction.

The proposed in-field repair strategy is summarized as follows:

- Apply self-test and repair in field
- Reconfigure the address mapping

  - Repair a word if it has 2 or more faulty bits. Remapped word with 1 faulty bit is cancelled if no more spare available.
  - Repair a word if it has 1 faulty bit and a spare is available. If no spare is available, do not repair the word and leave it to be corrected by ECC.

Spare words are included when test is performed to all words in memory. If a spare word is found to be faulty, a flag bit is used to set that the spare word is excluded from words that can be used for repair [4].

2.3 Remap Controller Implementation

To realize the in-field repair strategy, we implement a remap controller that executes an adaptive reconfiguration method. The remap CAM is divided into two areas based on fault classification and we use two address counters RC2F and RC1F. The top part is used for uncorrectable words and the bottom part is used for correctable words. The counter RC2F points to the next address to be written into for uncorrectable words in the top part, and the counter RC1F points to the next address to be written into for correctable words in the bottom part. When there are enough unused spare words, all of the faulty words are repaired, wherein remap CAM is adequately maintained according to the in-field repair strategy, as explained in the previous section. When remap CAM is full, all of the spare words have been used for repair. After this, only uncorrectable words are added to the remap CAM. This is achieved by removing previously remapped correctable words.

Figures 2 and 3 show examples on how to reconfigure the remap CAM where we assume both RC2F and RC1F point to a valid spare memory word. Figure 2 shows a case where the remap CAM has enough unused spare words, and a new faulty word is written into the remap CAM. Since the word is a new entry, the address is not yet stored in the remap CAM. In a case where a new entry is an uncorrectable word, the address is written at the location pointed to by RC2F, and then RC2F is decremented. In a case of correctable word, its address is written at the location pointed to by RC1F, and then RC1F is incremented. Figure 3 shows a case where remap CAM has no more spare words and new faulty word is found. In a case where a new entry is an uncorrectable word, the address is written at the location pointed to by RC2F, and then RC2F is decremented. Since an uncorrectable word is given priority in our in-field repair strategy, the uncorrectable word address is overwritten into the entry which previously stored a correctable word address. On the other hand, if a new entry is a correctable word address, no action is done since priority is given to uncorrectable word addresses.

Table 1 shows all the possible cases for remap CAM operation. The left column shows 2 types of faulty word address entry. The remap controller classifies the faulty word address to be written into the remap CAM as either a new entry or an old entry. When a faulty word address is compared to the contents of the remap CAM, and if it gives a miss, this means it is not yet stored at the remap CAM and the entry is new. If it gives a hit, it is denoted as a hit address, which means it is already stored at the remap CAM, and the entry is old. The middle column shows the number of faulty bits in a self-test period. Besides executing the in-field repair strategy for new entries, the remap controller considers whether the number of faulty bits varies or remains the same in a self-test period for old entries, and executes as such the corresponding operations. The operations write, increment and decrement take address pointers as parameters, where write means storing a faulty word address at the address in remap CAM pointed by the address pointer, while increment and decrement means to increment or decrement values of the address pointer. For the copy operation, we copy contents pointed by an address pointer (left operand) to a destination address (right operand). For the swap operation, we swap the contents pointed by the address pointers. The right column shows remap controller operations depending on the number of stored entries of the remap CAM. The remap controller executes operations whether a) the remap CAM has enough space to write a new entry, b) the remap CAM is full composed of both uncorrectable and correctable word addresses, or c) the remap CAM is full of uncorrectable word addresses. Memory fails when the remap CAM is full with uncorrectable words, and a new entry occurs.

3. Reliability Model

In this section, we explain the stochastic model used in order
to evaluate the reliability of the proposed strategy. Also, we describe the environment under which this evaluation is performed, specifically the periodicity of the time when self-test and repair, as well as memory scrubbing, is applied.

Reliability of the proposed strategy is evaluated under assumption that hard and soft errors occur with Poisson distributions. Let $\lambda_h$ and $\lambda_s$ be error rates of a single bit for hard and soft errors, respectively. Let $N$ and $N_s$ denote the number of words and spare words, respectively. The memory system provides $N$ words to users among $N + N_s$ words.

Reliability of a memory is evaluated in the interval $[0, t]$. This is the probability that a system is working well during $[0, t]$, which means the system has no uncorrectable word during $[0, t]$. Periodic self-test and repair and periodic scrubbing protect memory from hard and soft errors. Since scrubbing is more frequently applied than self-test, the interval $[0, t]$ is divided into several self-test periods and one self-test period is further divided into scrubbing periods as shown in Fig. 4. Let $t_s$ and $t_r$ denote the lengths of self-test period (an interval between two successive self-tests) and scrubbing period (an interval between two successive scrubblings), respectively. The system is reliable if all the past and current self-test periods are reliable, and one self-test period is reliable if all the scrubbing periods in the self-test period are reliable. At any given moment $t$, a system works well, or has no uncorrectable word, if every user word has 1) no error, 2) one hard error and no soft error, or 3) no hard error and one soft error.

The Poisson distribution gives the probability that no hard error occurs for one word during $[0, t]$ as

$$P_{h0}(t) = e^{-\lambda_ht},$$

where $n$ is the number of bits in a codeword, the probability that $n − 1$ bits have no hard error is given as

$$P_{h0}(n−1) = e^{-\lambda_ht} \cdot e^{-\lambda_ht},$$

and the probability that exactly one bit gets hard error during period $[0, t]$ is given as

$$P_{h1}(t) = n \left(1 - e^{-\lambda_ht} \right) \cdot e^{-\lambda_ht}.$$

The probability that 2 or more bits get hard errors during period $[0, t]$ is given as

$$P_{>h1}(t) = 1 - P_{h0}(t) - P_{h1}(t),$$

and the probability that 1 or more bits get hard errors during period $[0, t]$ is given as

$$P_{>h}(t) = 1 - P_{h0}(t).$$

The probability that no soft error occurs during period $[0, t]$ is given as

$$P_{s0}(t) = e^{-\lambda_st},$$

the probability that $n − 1$ bits have no soft error is given as

$$P_{s0}(n−1) = e^{-\lambda_st} \cdot e^{-\lambda_st},$$

and the probability that exactly one bit gets a soft error during period $[0, t]$ is given as

$$P_{s1}(t) = n \left(1 - e^{-\lambda_st} \right) \cdot e^{-\lambda_st}.$$

Assume that a word has no hard error during $[0, t]$. During this interval, the word works well if the word does not have multiple soft errors at the same time. Since scrubbing eliminates soft errors if each word has at most one soft error, the probability that at most one bit has soft error at the same time during $[0, t]$ if there is no soft error at 0 is given as

$$P_{s\leq1}(t) = \left[ (e^{-\lambda_st} + n \left(1 - e^{-\lambda_st} \right)) \cdot e^{-\lambda_st} \cdot (e^{-\lambda_st} + n \left(1 - e^{-\lambda_st} \right)) \right] \cdot e^{-\lambda_st} = n \left(1 - e^{-\lambda_st} \right) \cdot e^{-\lambda_st},$$

where $c = \lfloor t/t_s \rfloor$ is the number of scrubbing in $[0, t]$, and $t_s = t - ct_s$ is the elapsed time after the last scrubbing.

The reliability for one word during $[0, t]$ within the 1st self-test period is given as

$$R^0_w(t) = P_{h0}(t) \cdot P_{s\leq1}(t) + P_{h1}(t) \cdot P_{s0}(n−1)(t),$$

where any word has no error at time 0 is assumed. Using this reliability, the reliability of memory system during $[0, t]$ within the 1st self-test period is given as

$$R^0(t) = \left( R^0_w(t) \right)^N.$$
This also expresses the reliability of a memory system that does not repair faulty words with one hard error.

The reliability of memory during the 1st period is given as

$$R_{period}(1) = R^0(t_e).$$

The reliability of one word during one test period for an interval $$[t, t + t_e]$$, where $$t$$ is the time at the beginning of the current self-test period and $$t_e$$ is the elapsed time in the current test period, is given as

$$R^0_{w}(t_e) = P_{h0}(t_e) \cdot P_{s\leq 1}(t_e) + P_{h1}(t_e) \cdot P_{e\leq-1}(t_e),$$

if there is no hard error at $$t$$, and

$$R^1_{w}(t_e) = P_{h0}^{e=1}(t_e) \cdot P_{e=0}(t_e),$$

if there is 1 hard error at $$t$$.

The memory system is still reliable if one of the following two conditions hold as a result of self-test: 1) there are at least $$N$$ words with 0 hard error, or 2) there are less than $$N$$ words with 0 hard errors, but at the same time, there are at least $$N$$ words with 0 or 1 hard errors.

For the condition that there are at least $$N$$ words with 0 hard error in the memory at the beginning of the $$k$$-th period, the probability is given as

$$P_0(k) = \sum_{i=j=N}^{N+N_i} \binom{N}{i} \left( P_{h0}(k-1, t_i) \right)^i \cdot \left( P_{e\leq-1}(k-1, t_i) \right)^{N+N_i-i},$$

where there are $$i$$ words with 0 hard errors, and $$N+N_i-i$$ words with 1 or more errors.

For the condition that there are $$i$$ words ($$i < N$$) with 0 hard error and at least $$N$$ words at most one hard error at the beginning of the $$k$$-th period can be expressed as

$$P_{0(i)+1}(k) = \sum_{j=i-N}^{N-N_i} \binom{N}{j} \left( P_{h0}(k-1, t_i) \right)^j \cdot \left( P_{h1}(k-1, t_i) \right)^{N-N_i-j},$$

where there are $$i$$ words with 0 hard errors, $$j$$ words with 1 hard errors, and $$N+N_i-j$$ words with 2 or more errors.

Then, for the condition that there are less than $$N$$ words with 0 hard errors and at least $$N$$ words with 0 or 1 hard errors at the beginning of the $$k$$-th period, the probability is given as

$$P_{0+1}(k) = \sum_{i=0}^{N-1} P_{0(i)+1}(k).$$

The reliability for the $$k$$-th period is given as

$$R_{period}(k) = \frac{P_0(k)}{P_0(k) + P_{0+1}(k)} \left( R^0_{w}(t_e) \right)^N$$

$$+ \sum_{i=0}^{N-1} \frac{P_{0(i)+1}(k)}{P_0(k) + P_{0+1}(k)} \left( R^0_{w}(t_e) \right)^i \left( R^1_{w}(t_e) \right)^{N-i}.$$

To complete the expression for the reliability as a function of time, the reliabilities for all the previous $$(k-1)$$-th periods are multiplied to the current period until the elapsed time, and the reliability of memory is given as

$$R(t) = \prod_{p=1}^{k-1} R_{period}(p) \times \left[ \frac{P_0(k)}{P_0(k) + P_{0+1}(k)} \left( R^0_{w}(t_e) \right)^N$$

$$+ \sum_{i=0}^{N-1} \frac{P_{0(i)+1}(k)}{P_0(k) + P_{0+1}(k)} \left( R^0_{w}(t_e) \right)^i \left( R^1_{w}(t_e) \right)^{N-i} \right]_t,$$

where $$t$$ is the time in the $$k$$-th period, and $$t_e = t \mod t_i$$ is the time elapsed after the last self-test.

4. Experimental Results

In this section, we present the evaluation results of the reliability model, and hardware overhead of the remap controller.

4.1 Reliability Evaluation

The reliability is evaluated using a data analytical and graphing software. Nominal values are chosen as baseline as shown in Table 2. The number of bits in a code word $$n$$ considers 16-bit memories and 5 redundant bits to provide error correction. Soft errors must occur much more frequently than hard errors, wherein $$\lambda_h \ll \lambda_s$$, since soft errors are transient phenomenon, while hard errors are physical defects that occur after continued usage of the device. The scrubbing interval $$t_s$$ is set to occur every 6 minutes, while self-test and repair is set to occur every 10 days. The reliability of the proposed strategy is compared to the reliability of the traditional combined approach strategy [2], [5], where only uncorrectable words are repaired, while correctable words are left to the ECC.

We know from the experts’ hearings that a part of the power plant products requires high reliability for lifetime longer than 20–50 years, though usual products and applications require reliability of 10 years for normal systems, and 20 years for automotive. Throughout 50 years, including the 10-year and 20-year marks, the results show that the proposed strategy has better reliability than the traditional strategy.

We vary parameters in order to observe how the proposed strategy improves reliability. Figure 6 shows the results of the reliability evaluation for various memory sizes. The reliability of the proposed strategy is close to 1 all throughout the observed range, except for the case of 512K words. Also, the proposed strategy is more reliable by several decades than the traditional strategy. Figure 7 shows the results for various spare word sizes. All the plots for the traditional strategy coincide with each other. This shows that even if there are enough spare words, if correctable

| Parameters | Definition | Values |
|------------|------------|--------|
| N          | number of words | 1e+5   |
| Ns         | number of spare words | 50     |
| n          | number of bits per word | 21     |
| \(\lambda_h\) | hard error rate (errors per hour) | 10^11  |
| \(\lambda_s\) | soft error rate (errors per hour) | 10^-7  |
| t_i        | time between self-test and repair (hours) | 240    |
| t_e        | time between scrubbing (hours) | 0.1    |
words are not repaired, the traditional strategy is not able to mitigate the errors, and system reliability is severely degraded. The plots of both 50 and 500 spare words for the proposed strategy coincide with each other. This shows that for a given baseline of error rate and for a given time period that the reliability of the system is observed, a minimum number of spare words can be enough to maintain reliability. Figure 8 shows that case when hard errors occur more often. Though the proposed strategy may use up all of the spare words within several decades, it has better reliability than the traditional strategy. Figure 9 shows the case when soft errors occur more often. The proposed strategy provides higher reliability than the traditional strategy, though the system reliability degrades quickly. For high soft error rates, the system might require stronger soft error protection. Figure 10 shows the results for various word lengths, where two numbers in the legend mean lengths of an original word and redundant bits for ECC. Memories with longer word lengths are more susceptible to errors, and the proposed strategy provides protection from these errors. Figure 11 shows the results for varying self-test and repair periods. The result shows that as long as the proposed strategy performs self-test and repair on the ample period needed, it will be able to protect the memory system. When self-test and repair is done in a slower rate, the reliability of the system is affected.
All in all, the proposed strategy shows enhanced reliability performance compared to the traditional strategy.

4.2 Hardware Overhead

We have implemented the remap controller of our proposed in-field repair strategy using the Synopsys SAED90nm_EDK library. Controllers were implemented for 5 types of memory sizes given a fixed number of words for spare memory, diagnosis and remap CAM. The size of each entry for the diagnosis and remap CAM depends on memory size. We evaluate the overhead introduced by the remap controller to determine the area cost. Also, we evaluate the power and latency overhead of diagnosis CAM and remap CAM, and present the method and tool used to perform this evaluation. Then, we evaluate the power and latency overhead of the periodic test strategy by assessing the number of operations needed to perform test. We also cite a method that can minimize power and latency.

Table 3 shows the values of memory elements for a case of $1e+5$ words. The remap CAM has the same number of entries as the spare words, but has twice the address width, since each entry consists of a faulty word address and the corresponding spare word address. The diagnosis CAM has 256 word entries with word size of 38 bits, 17 bits for the faulty word address and 21 bits, which is the memory word size, since these are where the faulty bit locations are stored. The rest of the memory sizes use an extra addressing bit to be able to map into the spare addresses. For example, a memory with 8K words is addressed with 13 bits to access the spare words, so in total, 14 bits are used.

Table 4 shows the experimental results gathered for increasing memory sizes, where areas are evaluated in 2-input NAND gate-equivalent. The size of the remap controller slightly increases as memory words increase. This is because the number of addressing bits logarithmically increases with increase in memory size. For all the types, the hardware overhead is less than 1,800 gates, which is small compared to the total number of gates for an SoC. It also shows that the remap controller is scalable.

We evaluate power and latency overhead using memory elements shown in Table 3. The power overhead is mainly contributed by the CAM operation due to the comparison circuitry activated in parallel. Both CAMs are relatively small in size. The bigger diagnosis CAM is only used during test mode, while the smaller remap CAM is used during test mode, and during user mode, where remapping is in effect. The CAM size limits the number of operations performed by the CAM. We use CACTI [12] to evaluate power and latency for 90nm technology. CACTI is used for cache memories, and does not fully support CAM evaluation. Nevertheless, we can use the tool by carefully choosing parameters that are close to that of the CAMs in Table 3. CACTI can evaluate set associative cache memories with at least 16 sets, associativity of a power of 2, and block size of at least 8 bytes. On the other hand, the CAMs in Table 3 require full associative (single set) caches with 256 entries (associativity) and 17-bit tag and 21-bit data (block size) for the diagnosis CAM, and 50 entries and 17-bit tag and 17-bit data for the remap CAM. We evaluated caches with 16 sets and 8-byte (64-bit) block, and 256 entries for the diagnosis CAM and 64 entries for the remap CAM. In this setup, each of the diagnosis CAM and the remap CAM is considered as one of the sets of a set associative cache. However, when accessing a cache, only one set becomes active, and we consider that we can evaluate power and latency of our CAMs with a slight overestimation. The results are shown in Table 5. We find the dynamic power and leakage power for each set of the CAMs to be relatively small. Also, we find the access time, and thus latency overhead, to be relatively small.

We also evaluate the power and latency overhead of periodic test strategy. We assess the memory BIST, which uses a March-like self-test for a word-based memory organization. The MATS+ [13] is used to perform self-test and has test time of $5 \times n/B$, where $n$ is number of cells and $B$ is number of bits in a word. Given that memory organization is composed of $N$ user words and $N_s$ spare words, the total time to perform memory test is $5 \times (N+N_s)$ operations, which are relatively few.

Also, we assess the number of times the CAMs may be accessed during periodic test and repair, which is dependent on the number of faults found. Given the memory in Table 3, for a hard error rate of $\lambda_b = 10^{-11}$, the faults that have occurred are expected to be $\sim 1.83$ at the 10-year mark, and $\sim 3.64$ at the 20-year mark, while for a hard error rate of $\lambda_b = 10^{-10}$, the faults that have occurred are expected to be $\sim 18.35$ at the 10-year mark, and $\sim 36.4$ at the 20-year mark. From this, we learn that at first, less faults occur, and the number of accesses to the CAMs are infrequent. Then after some time, more faults occur, and the number of accesses to the CAMs increase. This implies that power or latency over-
5. Conclusion

Given an ECC-based memory architecture, a novel in-field repair strategy that repairs uncorrectable words and possibly correctable words is proposed. An adaptive reconfiguring method is executed by the implemented remap controller to realize the proposed in-field repair strategy. A stochastic reliability model of the proposed strategy is established based on Poisson distributions for hard and soft errors. The proposed strategy enhances memory reliability in field compared to the traditional strategy, which repairs uncorrectable words only. Also, the hardware overhead of area, power and latency is found to be small. These show that the proposed strategy extends memory lifetime in practical use.

References

[1] E.J. Marinissen, B. Prince, D. Keitel-Schulz, and Y. Zorian, “Challenges in embedded memory design and test,” Proc. DATE’05, vol.2 pp.722–727, March 2005.

[2] T.-H. Wu, P.-Y. Chen, M. Lee, B.-Y. Lin, C.-W. Wu, C.-H. Tien, H.-C. Lin, H. Chen, C.-N. Peng, and M.-J. Wang, “A memory yield improvement scheme combining built-in self-repair and error correction codes,” Proc. ITC 2012, pp.1–9, no.2012.

[3] M. Horiguchi, K. Itoh, Nanoscale Memory Repair, Springer, New York, NY, USA, 2011.

[4] M. Nicolaidis, N. Achouri, and L. Anghel, “A diversified memory built-in self-repair approach for nanotechnologies,” Proc. IEEE VTS 2004, pp.313–318, April 2004.

[5] M. Nicolaidis and P. Papavramidou, “Transparent BIST for ECC-based memory repair,” Proc. IEEE IOLTS 2013, pp.216–213, July 2013.

[6] C.L. Su, Y.-T. Yeh, and C.-W. Wu, “An integrated ECC and redundancy repair scheme for memory reliability enhancement,” Proc. IEEE DFT 2005, pp.81–89, Oct. 2005.

[7] J.F. Li, J.-C. Yeh, R.-F. Huang, and C.-W. Wu, “A built-in self-repair scheme for semiconductor memories with 2-D redundancy,” Proc. ITC 2003, pp.393–402, Oct. 2003.

[8] A. Benso, S. Chiusano, G.D. Natale, and P. Prinetti, “An on-line BIST RAM architecture with self-repair capabilities,” IEEE Trans. on Rel., vol.51, no.1, pp.123–128, March 2002.

[9] B. Schroeder, E. Pinheiro, and W.-D. Weber, “DRAM errors in the wild: a large-scale field study,” Proc. SIGMETRICS ’09, pp.193–204, June 2009.

[10] A. Saleh, J.J. Serrano, and J.H. Patel, “Reliability of scrubbing recovery-techniques for memory systems,” IEEE Trans. on Rel., vol.39, no.1, pp.114–122, April 1990.

[11] B. Jacob, S. Ng, and D. Wang, Memory Systems: Cache, DRAM, Disk, Morgan Kaufmann Publishers, San Francisco, CA, USA, 2007.

[12] http://quid.hpl.hp.com:9081/cacti/

[13] A.J. Van De Goor, I.B.S. Tili, and S. Hamdioui, “Converting march tests for bit-oriented memories into tests for word-oriented memories,” Proc. Int’l Workshop on MTDT 1998, pp.46–52, Aug. 1998.

[14] G. Mayuga, Y. Yamato, T. Yoneda, Y. Sato, and M. Inoue, “An ECC-based memory architecture with online self-repair capabilities for reliability enhancement,” Proc. IEEE ETS 2015, pp.1–6, May 2015.