Efficient Online and Offline Testing of Embedded DRAMs

Sybille Hellebrand, Hans-Joachim Wunderlich, Alexander A. Ivaniuk,
Yuri V. Klimets, and Vyacheslav N. Yarmolik

Abstract—This paper presents an integrated approach for both built-in online and offline testing of embedded DRAMs. It is based on a new technique for output data compression which offers the same benefits as signature analysis during offline test, but also supports efficient online consistency checking. The initial fault-free memory contents are compressed to a reference characteristic and compared to test characteristics periodically. The reference characteristic depends on the memory contents, but unlike similar characteristics based on signature analysis, it can be easily updated concurrently with WRITE operations. This way, changes in memory do not require a time consuming recomputation. The respective test characteristics can be efficiently computed during the periodic refresh operations of the dynamic RAM. Experiments show that the proposed technique significantly reduces the time between the occurrence of an error and its detection (error detection latency). Compared to error detecting codes (EDC) it also achieves a significantly higher error coverage at lower hardware costs. Therefore, it perfectly complements standard online checking approaches relying on EDC, where the concurrent detection of certain types of errors is guaranteed, but only during READ operations accessing the erroneous data.

Index Terms—Embedded memories, systems-on-a-chip, online checking, BIST.

1 Introduction

Present day systems-on-a-chip integrate a variety of different components, like processor cores, RAMs, ROMs and user-defined logic on a single chip. Growing integration densities have made it feasible to embed dynamic RAM cores of considerable sizes [25]. Embedded DRAMs offer a large degree of architectural freedom concerning the memory size and organization. Therefore, they are of particular interest for applications where high interface bandwidths have to be achieved, as, for example, in network switching. On the other hand, due the limited external access, testing embedded DRAMs is an even more challenging problem than testing monolithic DRAM chips. Here, a number of built-in self-test approaches which have been proposed in the literature can help to develop solutions [1], [2], [4], [5], [6], [7], [8], [10], [14], [15], [16], [18], [19], [21], [23], [28]. A typical BIST architecture is shown in Fig. 1.

The test pattern generator, for example, an LFSR or a counter, activates a sequence of addresses and, depending on the type of test, the test control unit initiates one or several operations on the respective memory cells. The resulting output data stream is fed into the data compressor, the final state of which provides a characteristic C_{TEST}. This is compared to a predetermined reference characteristic C_{REF}, and differences between both characteristics indicate the presence of faults. With increasing memory densities, the relative area for the BIST resources becomes negligible.

To deal with soft errors during system operation, adding standard online checking capabilities based on error detecting codes (EDC) is the first step also for embedded DRAMs [22]. Depending on the specific code, the detection of certain types of errors can be guaranteed. But, since error detection is only possible during READ operations, the time between the occurrence of an error and its detection, referred to as error detection latency, may be very high. For some applications with high reliability requirements, e.g., in telecommunication switching, it is not acceptable to detect erroneous data only at the moment when the data are explicitly needed [3]. In contrast, errors should be detected as early as possible to allow for recovery before the data are requested by the system. Furthermore, EDCs have to increase the number of check bits to reduce the probability of masking multiple errors, which results in a high hardware overhead.

As a low-cost alternative, the BIST architecture of Fig. 1 can be reused for online consistency checking as follows: The pattern generator cycles through all possible addresses once and, at each address, the memory contents are read out and fed into the data compressor. This way a reference characteristic C_{REF} is "learned" and can be periodically compared to a test characteristic C_{TEST} computed in the same way as C_{REF}, but concurrently with the memory operation. There is no hardware overhead for storing check bits and the probability of error masking only depends on the properties of the data compressor. A 32-bit signature analyzer, for example, keeps the probability of masking

- *S. Hellebrand is with the Institute of Computer Science, University of Innsbruck, Technikerstr. 25, 6020 Innsbruck, Austria. E-mail: sybille.hellebrand@uibk.ac.at.*
- *H.-J. Wunderlich is with the Division of Computer Architecture, University of Stuttgart, Breitwiesenstr. 20-22, 70565 Stuttgart, Germany. E-mail: wu@informatik.uni-stuttgart.de.*
- *A. Ivaniuk, Y. Klimets, and V.N. Yarmolik are with the Computer Systems Department, Belararussian State University of Informatics and Radioelectronic, P. Brovki 6, 220027 Minsk, Belarus. E-mail: {ivaniuk, klimets}@bsuir.unibel.by, yarmolik@gw.bsuir.unibel.by.*

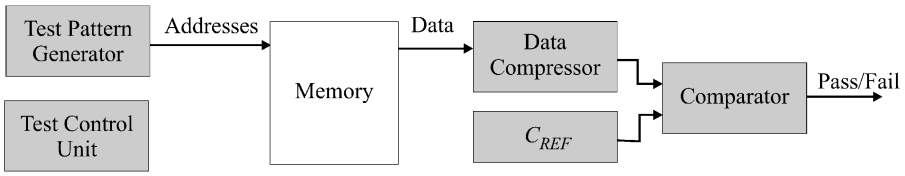

Fig. 1. Typical BIST architecture for memories.

arbitrary errors below 2^{-32}, which cannot be guaranteed by an error detecting code with a feasible number of check bits. However, to make this idea really working two problems have to be solved first.

- The reference characteristic depends on the memory contents and changes in memory also change the reference characteristic. If, for example, a conventional signature analyzer were used as output data compressor, then the initial learning phase would have to be repeated after every WRITE operation [20]. Therefore, an alternative technique for output data compression is required which allows for a fast and simple update of the reference characteristic concurrently with WRITE operations.
- To guarantee a low error detection latency, the test characteristics have to be computed with a high frequency, but the regular memory operation should not be interrupted or disturbed by this process.

In order to analyze the second problem in more detail, the basic logic structure of a dynamic RAM is recalled in the sequel. To avoid data retention, dynamic RAMs refresh data during READ/WRITE operations and during periodic refresh operations. In a typical memory organization, as shown in Fig. 2, the address is split into a row and a column address, and READ/WRITE operations first transfer the complete memory row indicated by the row address to the refreshment register (activated by the row access strobe RAS). The actual READ/WRITE operations are then performed on the refreshment register (activated by the column access strobe CAS) before its contents is written back to memory.

While the circuit level implementation of this structure may be distributed or scrambled and the refreshment register may be substituted just by amplifiers, the essential signals are available in most of the proprietary implementations and will be used in the rest of this paper.

The periodic refresh operations consist of transferring all memory rows to the refreshment register and loading them back to memory. Since the complete memory is scanned during a periodic refresh operation, this phase naturally offers itself for concurrently computing a test characteristic C_{TEST} as proposed above [12]. In contrast to more general schemes for the concurrent testing of digital circuits, it is guaranteed that all necessary test inputs actually appear during a test phase [24]. However, in contrast to an offline BIST implementation, it must be guaranteed that the computation can be completed within the time slot available for the periodic refresh operation. Furthermore, the algorithms for refreshment and for consistency checking should have a high degree of similarity to simplify control. Consequently, a characteristic which can be built step by step from row-characteristics is targeted. The time to compute a row characteristic must not exceed the time to refresh a row.

In this paper, a BIST architecture for embedded DRAMs is proposed which also solves the problems stated above and therefore provides a unique solution for both offline manufacturing and maintenance test and online consistency checking. It is based on the "modulo-2 address characteristic" introduced in [26], which is self-adjusting, i.e., after WRITE operations C_{REF} can be adjusted in one step. Before the necessary extensions for an efficient online computation of this characteristic are described in Section 3, its basic properties are briefly reviewed in Section 2. The complete online and offline testable memory architecture is presented in Section 4. As it will be shown in Section 2, for offline BIST, this architecture provides the same quality as conventional BIST schemes relying on signature analysis for output data compression. To evaluate its capabilities with respect to online consistency checking, experiments have been performed relying both on random simulations and on the simulation of real program data. The results documented in Section 5 show that the proposed approach combines a high error detection rate with a low error detection latency.

2 Self-Adjusting Output Data Compression

2.1 Basic Principles and Facts

In this section, the basic principles and properties of the modulo-2 address characteristic are briefly reviewed. As

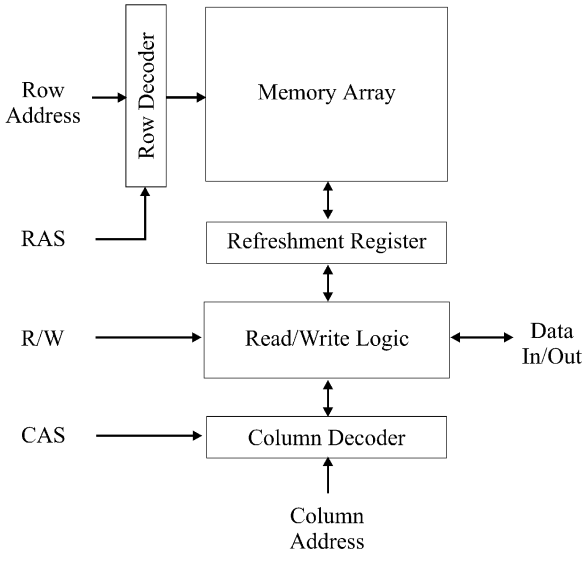

Fig. 2. Typical organization of a dynamic RAM.

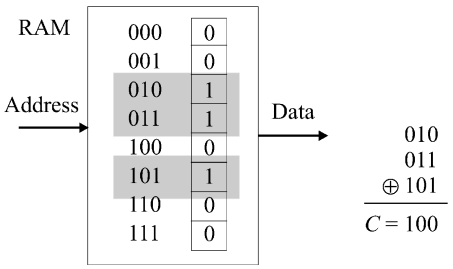

Fig. 3. Modulo-2 address characterisitic for bit-oriented RAMs.

shown in Fig. 3, the characteristic is obtained as the bit-wise modulo-2 sum of all addresses pointing to "1."

The characteristic allows for implementing periodic offline consistency checking in an efficient way because it can be easily adjusted concurrently with changes in the memory contents. In case of a WRITE operation at a specific address a, the old reference characteristic C_{REF}^{old} is updated to

$$C_{REF}^{new} = C_{REF}^{old} \oplus a \cdot \Big(M[a]^{new} \oplus M[a]^{old} \Big),$$

where $M[a]$ denotes the memory contents at address a.

For computing the complete characteristic, as well as for updating it concurrently with WRITE operations, a simple compressor circuit can be used which performs bit-wise EXOR operations on the address lines controlled by the data input. To calculate the initial characteristic C_{REF} or the test characteristic C_{TEST}, a counter or an LFSR has to generate all memory addresses. If the memory operation starts after a reset to zero, C_{REF} is known to be zero and the initialization can be skipped. The basic architecture of the complete memory with built-in consistency checking is shown in Fig. 4.

In [26], it has been shown that relying on the modulo-2 address characteristic achieves the same quality as characterizing the memory by conventional signature analysis:

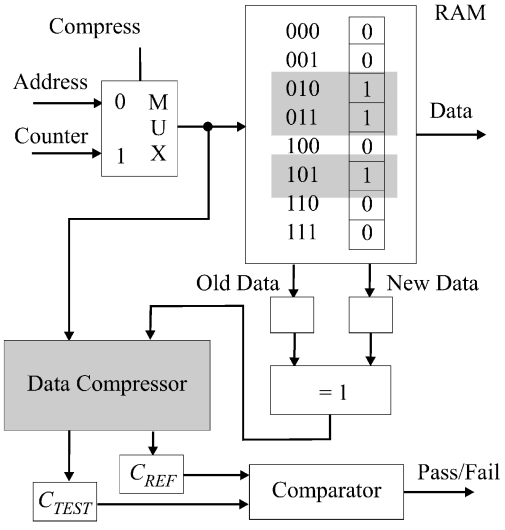

Fig. 4. Consistency checking based on the modulo-2 address characteristic.

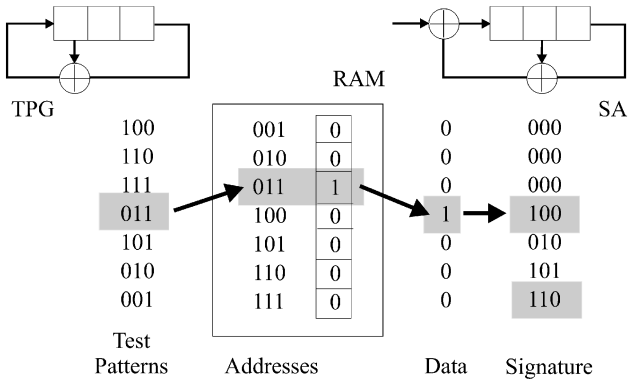

Fig. 5. Correspondence between signature analysis and modulo-2 address characteristic.

1. All single errors are detectable and diagnosable. If only single errors are assumed, the expression, $C_{REF} \oplus C_{TEST}$, provides the address of the faulty memory cell.

   However, in the basic scheme of Fig. 4, the memory address $(0, \ldots, 0)$ does not contribute to the characteristic and therefore errors in this memory location would not be detected. A simple work-around to avoid this problem in a practical implementation is to add an extra bit with a constant "1" to all memory addresses. Other solutions are possible.
2. All double errors are detectable since, in this case, $C_{REF} \oplus C_{TEST}$ corresponds to the sum of two addresses a_r and a_s, and $a_r \neq a_s$ implies $C_{REF} \oplus C_{TEST} \neq 0$.
3. Data compression based on the modulo-2 address characteristic is equivalent to serial signature analysis and the probability of aliasing errors is thus estimated by 2^{-k}, where k denotes the length of the characteristic.

Property 3 is an immediate consequence of the following observation.

Observation. *Let $\varphi(X) \in \mathrm{GF}(2)[X]$ be a primitive polynomial of degree k, and let $\varphi^{-1}(X) := X^k \varphi(\frac{1}{X})$ denote the reciprocal polynomial. An LFSR with feedback polynomial $\varphi^{-1}(X)$ and initial state $(1, 0, \ldots, 0)$ generates the same state transition sequence (in reverse component order) as the LFSR with feedback polynomial $\varphi(X)$ "counting" backward from $(0, \ldots, 0, 1)$.*

The example shown in Fig. 5 exploits this observation to verify property 3 for a 7-bit RAM. A conventional BIST is implemented using a 3-bit LFSR with primitive feedback polynomial $\varphi(X) = 1 + X + X^3$ as test pattern generator and a serial signature analyzer with the reciprocal feedback polynomial $\varphi^{-1}(X) = 1 + X^2 + X^3$.

With an all-zero initial state, the signature register does not change its contents before the first cell containing a "1" is addressed. The new contents is (1, 0, 0), and as the remaining memory cells only contain "0" entries, the signature analyzer works like an autonomous LFSR with initial state (1, 0, 0) for the rest of the test procedure. Since $\varphi^{-1}(X)$ is the reciprocal of $\varphi(X)$, this implies that the

signature analyzer basically behaves like the test pattern generator counting backward from (0, 0, 1) (with reversed component order), thus the final signature is (1, 1, 0) and corresponds exactly to the address of the memory cell with a "1" entry.

If the RAM contains more than one nonzero entries, then similarly the signature is obtained as modulo-2 sum of all addresses (in reverse component order) corresponding to memory cells with contents "1."

In general, the correspondence between the modulo-2 address characteristic and signature analysis is described by the following theorem which was proven in [14], [27].

Theorem 1. *Let M be a bit-oriented memory with $m = 2^k - 1$ cells, $\varphi(X) \in \mathrm{GF}(2)[X]$ a primitive polynomial of degree k, and let $A_1 \subset \mathrm{GF}(2)^k \setminus \{0\}$ contain the memory addresses pointing to "1" entries. Furthermore, for $a = (a_0, \ldots, a_{k-1}) \in \mathrm{GF}(2)^k$ let $a^r := (a_{k-1}, \ldots, a_0)$ denote the vector with components in reverse order. Then, a BIST*

- *using a test pattern generator with feedback polynomial $\varphi(X)$,*
- *a serial signature analyzer with feedback polynomial $\varphi^{-1}(X)$,*
- *initial states $(1, 0, \ldots, 0)$ and $(0, \ldots, 0)$ for the test pattern generator and the signature analyzer, respectively,*
- *and a test length of m*

is characterized by the fault-free signature $S = \oplus_{a \in A_1} a^r$.

The theorem remains true when the number of memory cells is m $< 2^k - 1$, and the initial state of the test pattern generator is selected, such that the final state is $(0, \ldots, 0, 1)$. This implies that, for any memory BIST based on the modulo-2 address characteristic, there exists an equivalent BIST configuration based on signature analysis with a primitive feedback polynomial and, consequently, the same test quality is guaranteed.

In contrast to conventional signature analysis, however, changes in memory do not require the time-consuming recomputation of C_{REF}. As shown above, adjusting the characteristic is simply achieved by

$$C_{REF}^{new} = C_{REF}^{old} \oplus a \cdot \left(M[a]^{new} \oplus M[a]^{old} \right).$$

For an efficient implementation, the comparison of the old and new memory contents is the crucial point, and extra READ operations to get the old contents should be avoided. In dynamic RAMs, the old memory contents is transferred to the refreshment register anyway, before it is overwritten. The memory architecture described in detail in Section 4 exploits this fact to integrate online consistency checking without extra READ operations or performance losses.

Since the only differences between the checking procedure described above and common built-in self test procedures are given by the complexity of the memory operations applied to each cell and the number of runs through the memory [11], [19], the proposed technique for output data compression can, of course, be used also during manufacturing and maintenance test.

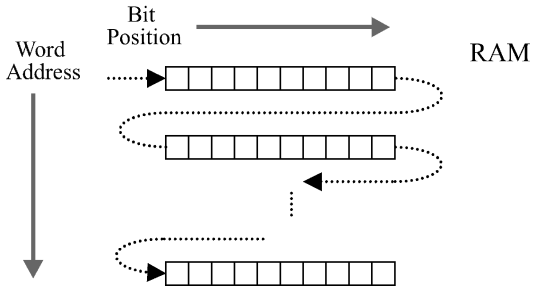

Fig. 6. Bit-oriented representation of a word-oriented RAM.

2.2 Extension to Word-Oriented RAMs

The scheme for output data compression introduced in the previous section can easily be applied to word-oriented RAMs [26] as illustrated in Fig. 6.

For this purpose, the word-oriented RAM is considered as bit-oriented memory with addresses of the form (a_w, a_b), where a_w denotes the word address and a_b the bit position within the word. The memory can then be modeled as a two-dimensional array $M[1 \ldots m, 0 \ldots n]$ of bits with address space $A = \{1, \ldots, m\} \times \{0, \ldots, n\}$ and "1"-space $A_1 := \{(a_w, a_b) \in A | M[a_w, a_b] = \text{"1"}\}$. The reference characteristic C_{REF} for the initial correct memory contents is determined as

$$C_{REF} = \bigoplus_{(a_w, a_b) \in A_1} (a_w, a_b),$$

where the modulo-2 sum of address pairs is defined by

$$(a_w, a_b) \oplus (a'_w, a'_b) = (a_w \oplus a'_w, a_b \oplus a'_b).$$

3 Online Consistency Checking

The basic technique described in Section 2 is not efficient enough to be applied during a periodic refresh operation because it steps through the memory bit by bit. Instead, a data compressor is required which is able to compute the partial characteristic corresponding to one row in one step. As illustrated in Fig. 7, the memory addresses are split into row and column addresses $a = (a_r, a_c)$. In the case of word-oriented memories, a column address can simply be considered as a vector of bit addresses, which does not change anything in the proposed techniques. For the sake of clarity, only the bit-oriented case is therefore considered in the sequel. If $A_1(r) := \{a_c | M[a_r, a_c] = 1\}$ denotes the set of all column addresses pointing to a "1" in row r, then the compressor must be able to determine

$$C_r = \bigoplus_{a_c \in A_1(r)} (a_r, a_c)$$

in one step. The complete characteristic is then obtained step by step as

$$C_{TEST} = \bigoplus_{0 \le a_r < m-1} C_r = \bigoplus_{0 \le a_r < m-1} \bigoplus_{a_c \in A_1(r)} (a_r, a_c),$$

where m denotes the number of rows.

The basic principle of such a generalized compressor is shown in Fig. 8.

The row characteristic

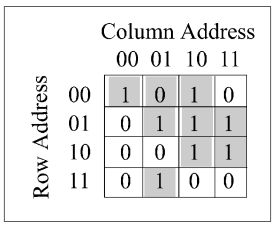

Fig. 7. Row-wise computation of the modulo-2 address characteristic.

$$C_r = \bigoplus_{a_c \in A_1(r)} (a_r, a_c) = \left(\bigoplus_{a_c \in A_1(r)} a_r, \bigoplus_{a_c \in A_1(r)} a_c \right)$$

has either a_r or 0 as its first component, depending on the parity of the memory row. For n columns, the second component is represented by a binary l-bit vector, $l = \lceil \log_2 n \rceil$. It is obtained by bit-wise EXOR-operations

$$\bigoplus_{a_c \in A_1(r)} a_c = \left(\bigoplus_{a_c \in A_1(r)} a_c^0, \ldots, \bigoplus_{a_c \in A_1(r)} a_c^{l-1} \right)$$

with a_c^i denoting the ith component of a_c, $0 \leq i < l$. As only bit-addresses with $a_c^i = 1$ can contribute to the ith sum and they only contribute the ith sum when the memory contains a "1," it is sufficient to implement functions F_i which count (modulo 2) the number of ones at all the addresses with $a_c^i = 1$. The second component of C_r is then derived as

$$\bigoplus_{a_c \in A_1(r)} a_c = (F_0, \ldots, F_{l-1}).$$

In the example of Fig. 8, the Function F_0 counts (modulo 2), the number of ones at bit position zero which can contribute to the characteristic, and F_1 does the same for bit position one. The parity function decides whether the row address contributes to the characteristic or not.

It can be easily verified that the EXOR tree, for implementing the parity check and the functions $F_0, \ldots, F_{l-1}$, requires at most

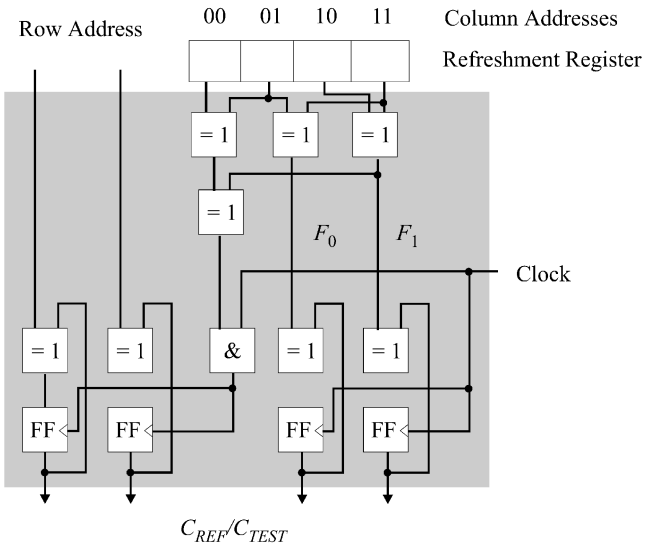

Fig. 8. Data compressor for the fast computation of row characteristics.

$$\sum_{1 \leq j \leq l} (2^j - 1) = 2^{l+1} - 2 - l = 2n - 2 - l$$

2-input EXOR-gates. Overall, the output data compressor of Fig. 8 can be implemented using $\lceil \log_2 m \rceil + l$ flip-flops, $\lceil \log_2 m \rceil + l + 2n - 2 - l = \lceil \log_2 m \rceil + 2n - 2$ EXOR-gates, and 1 AND-gate.

The time required to compute a row characteristic C_r is mainly determined by the depth of the AND/EXOR network between the refreshment register and the register containing the address characteristic. Using only 2-input gates, there are $l - 2$ levels in the EXOR tree for the parity check and the functions $F_0, \ldots, F_{l-1}$. The depth of the network is $l - 1$ in this case, it can be further reduced if gates with more than two inputs are employed.

Assuming a $1,024 \times 1,024$ bit dynamic memory, the data compressor can be, for example, implemented with 20 flip-flops, 2,056 2-input EXOR gates, and one AND gate. The delay through the AND/EXOR network corresponds to $9 \cdot d$, where d is the delay of one EXOR gate. If, furthermore, a row access time of 100 ns is assumed, then a gate delay $d < 100/9 \approx 11$ ns is sufficient enough to compute the row characteristic concurrently with the refreshment of the row and the complete characteristic C_{TEST} within the time slot for a periodic refresh operation [17].

4 The Complete Memory Architecture

This section briefly sketches the complete architecture of an embedded DRAM with BIST and error detecting refreshment. The core of the logic structure, shown in Fig. 9, is the generalized data compressor described in Section 3, which is used for both offline BIST and online consistency checking.

During online consistency checking, in case of a WRITE operation, C_{REF} has to be updated when the old and new memory contents differ. Rewriting the corresponding formula

$$C_{REF}^{new} = C_{REF}^{old} \oplus a \cdot \left(M[a]^{new} \oplus M[a]^{old} \right)$$

to

$$C_{REF}^{new} = C_{REF}^{old} \oplus a \cdot M[a]^{old} \oplus a \cdot M[a]^{new} \qquad (*)$$

provides a very simple and efficient architecture for DRAMs with BIST and error detecting refreshment.

If the memory operation does not start with a reset ($C_{REF} = 0$), a row counter, which cycles through all states, is sufficient to determine the initial characteristic. C_{REF} can be computed from the row characteristics as described above. During normal memory operation, WRITE requests

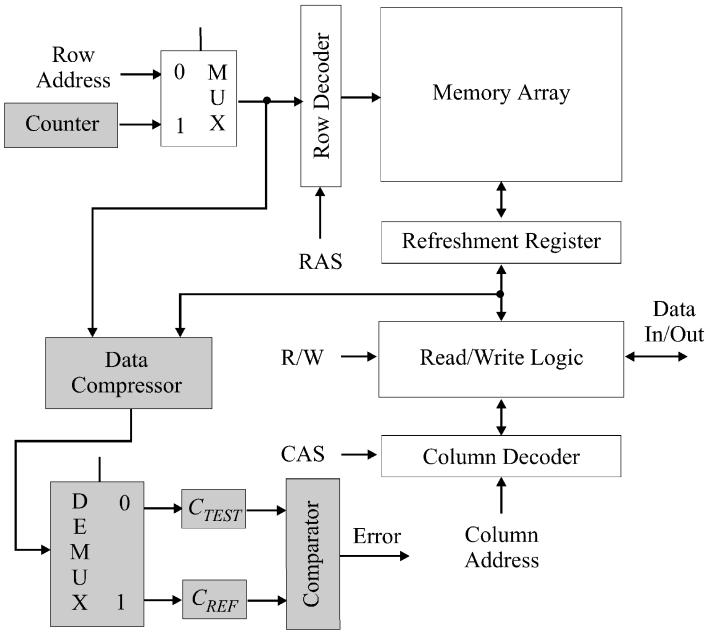

Fig. 9. Complete architecture for a DRAM with error detecting refreshment.

initiate a concurrent update of C_{REF}. In the first phase, the old contents of the memory row are transferred to the refreshment register and fed into the data compressor. In the second phase, the actual WRITE operation is performed on the refreshment register which is again fed into the data compressor, thus adjusting C_{REF} according to (*). Finally, during periodic refresh operations, the row counter enumerates all row addresses. Each row is transferred to the refreshment register and fed into the data compressor, which computes C_{TEST} as described in detail above. If a WRITE occurs at address $a = (a_r, a_c)$ during the periodic refresh operation, then C_{REF} must be updated as described above. Concerning C_{TEST}, the control unit has to distinguish between two cases:

- If the refresh procedure is interrupted at a row-address $a_r^* > a_r$, then C_{TEST} has to be adjusted, too, because the row characteristic for a_r has already been added to C_{TEST} before the WRITE operation at $a = (a_r, a_c)$.
- If the refresh procedure is interrupted at a row-address $a_r^* < a_r$, then the row characteristic for a_r has not yet been added to C_{TEST}, and there is no need to adjust C_{TEST}.

Since the row counter is required anyway to implement the periodic refreshment, the hardware overhead is mainly determined by the data compressor, the registers for C_{TEST} and C_{REF}, and the comparator (shaded blocks in Fig. 9). With the figures given above, this is negligible compared to the overall area of the memory. Compared to EDCs, the fault detection latency is reduced. With respect to the expected fault coverage in relation to the hardware cost, a comparison to EDC provides the following observation: Considering arbitrary multiple faults, the fault coverage of EDCs increases with $1 - \frac{2^n-1}{2^{n+k}-1}$ for k additional columns added to the memory while the area overhead increases with $O(k \cdot m)$, where m is the number of rows and n is the number of columns. The additional hardware for the proposed approach increases only with $O(log_2 m + 2n)$, but obtains a fault coverage of $1 - 2^{-\log_2 m - \log_2 n}$, where n is the number of columns and m denotes again the number of rows of the memory. The presented approach is thus superior to EDCs in terms of fault coverage, fault latency, and area overhead, but it does not provide complete concurrency, like EDCs, as errors may also occur in the short phase between signature checking and data access, which will be detected only in the next checking phase.

Concerning the memory performance, the combined online and offline detection scheme based on the modulo-2-address characteristic does not increase access times since all test operations are performed concurrently with the refresh operations or when the memory is offline. Similarly, as for a standard memory BIST, the test hardware is not in the data path for the regular memory operation. Furthermore, in contrast to schemes relying on error detecting/correcting codes, the memory contents is unchanged and the time penalty for calculating the check information during a memory access is avoided. A slight performance impact may be due to the increased load being driven by the refreshment register. If this results in an actual performance loss, it depends on the implementation and an exact evaluation goes beyond the scope of this paper.

5 Experimental Evaluation

To evaluate the proposed scheme with respect to its capabilities for online error detection, it was compared to a standard online checking approach relying on parity codes. As pointed out earlier, in the standard approach, errors can only be detected during READ operations. But, on the other hand, if only single errors in memory words or rows are considered, then adding a simple parity bit to each

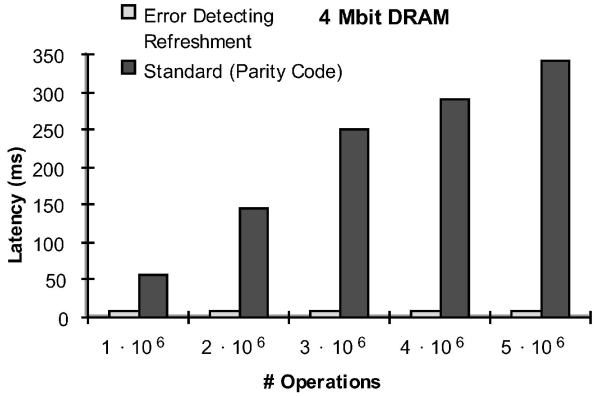

Fig. 10. Average error detection latency for a 4 Mbit DRAM.

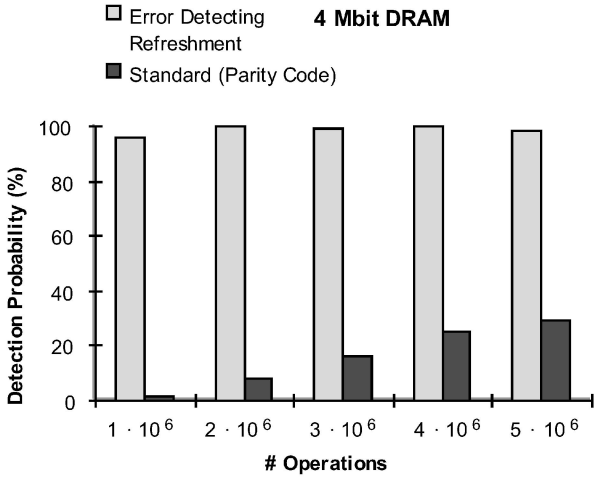

Fig. 11. Average detection probability for a 4 Mbit DRAM.

word or row already guarantees complete error detection, which, of course, is paid by a high hardware overhead. For the new scheme, a lower detection latency is expected on average; however, erroneous data may be reused before the error is detected. To characterize both methods more precisely, random simulations have been performed as well as simulations of the memory traffic for a set of benchmark programs. In all experiments, the following technology features were assumed for the DRAM [17]:

- Average access time for READ/WRITE operations: 200 ns.
- Refresh Period (time between two periodic refresh operations): 16 ms.
- Refresh time: $m \cdot 100$ ns (m denotes the number of rows in the memory array, and 100 ns is the row access time).

Concerning the error model, it was assumed that hard errors were detected during manufacturing test and that only soft errors had to be considered. The investigations focused on "**s**ingle **e**vent **u**psets" (SEUs) [9].

The first series of experiments was dedicated to random simulations according to the following setup:

- Sequences of 1 M to 5 M random operations at random addresses were simulated. The probability for both READ and WRITE operations was set to 0.5 and addresses were assumed to be uniformly distributed, too.
- DRAMs with a capacity from 1 Mbit to 4 Mbit were considered. For all experiments, a square organization of the DRAM was assumed, i.e., the number of rows varied from 1 K to 2 K.
- Single errors were injected at random time steps and at random addresses according to the uniform distribution.

For each combination of the parameters (length of the sequence, capacity of the memory), 100 simulations were performed with varying seeds for the random processes. The results showed the same basic trends for all memory sizes. Therefore, only the average detection latency and the fault detection probability for the 4 Mbit DRAM are reported in Figs. 10 and 11.

Concerning the detection latency, the following trends could be observed:

- On the average, an error is detected after 50 percent of a refresh period so that the average error detection latency for the proposed technique is around 8 ms.
- The error detection latency for the standard approach relying on parity codes is considerably higher and increases with the length of the random sequences. In the best case, it is about 6 times higher than for error detecting refreshment and in the worst case about 40 times.

With respect to the fault coverage, the proposed technique achieved a fault coverage close to 100 percent in all experiments. Only in rare cases were errors masked by WRITE operations before the next periodic refresh operation. In contrast, the fault coverage for the standard approach increases with the length of the random sequences, but never exceeds 60 percent. In the worst case, it is even below 10 percent. For EDCs, the errors remained undetected because the corresponding data were not requested by the random sequence, and all test sequences still provided the correct results. However, it should be noted that the correct results could only be guaranteed because the simulations were restricted to single errors, and because in the case of EDCs each memory location was assumed to have its own parity bit. For the considered bit-oriented memories, this already implied that the memory area was twice as large as for the original memory. In the general case, when also multiple errors are considered, a complete error detection can no longer be guaranteed for EDCs.

Since for real application programs a uniform distribution of READ/WRITE accesses cannot be expected, a second series of experiments was carried out for the benchmark programs SPICE, TeX, and the GNU C-compiler GCC. The memory traces were produced by the cache simulator DINERO for the DLX processor [13]. The technology data for the RAM and the mechanism for fault injection were the same as for the random simulations. The results with respect to error detection latency and fault coverage are presented in Tables 1 and 2.

For error detecting refreshments, similar results are observed as described for the random experiments. For the standard approach based on parity codes, however, considerably higher latencies and lower fault coverages are

TABLE 1
Error Detection Latency for the
Benchmark Programs SPICE, TeX, and GCC

Benchmark	Error Detecting Refreshment	Standard
SPICE	8.06 ms	1061.0 ms
TeX	8.16 ms	240.4 ms
GCC	8.16 ms	921.0 ms

TABLE 2
Error Coverage for the
Benchmark Programs SPICE, TeX, and GCC

Benchmark	Error Detecting Refreshment	Standard
SPICE	100 %	1 %
TeX	100 %	0.5 %
GCC	100 %	2 %

obtained. This is due to the fact that in the benchmark programs the number of READ operations is considerably lower than in the random experiments, and errors are detected only during READ operations. With the new approach, errors in unused data were detected, too.

Considering both random and benchmark experiments, the results show that error detecting refreshment and standard online checking complement each other in an ideal way. On the average, error detecting refreshment provides a low error detection latency and only a few errors escape. This supports an early activation of low-cost recovery schemes. If the concurrent detection of certain types of errors has to be guaranteed, this can still be achieved by using appropriate EDCs. However, in general, the trade-off between cost and fault coverage is much better for the presented technique.

6 Conclusions

A new technique for online consistency checking of embedded DRAMs, error detecting refreshment, has been presented. It is based on the modulo-2 address characteristic, which can be computed efficiently within the time slots reserved for a periodic refresh operation. At little extra hardware cost, the technique guarantees low error detection latencies and high error coverages. Depending on the reliability standards to be achieved, it can complement or replace conventional online checking schemes based on error detecting codes, where the concurrent detection of certain types of errors is guaranteed, but high detection latencies and high hardware overhead must be expected.

References

[1] V.C. Alves, M. Nicolaidis, P. Lestrat, and B. Courtois, "Built-in Self-Test for Multi-Port RAMs," *Proc. IEEE Int'l Conf. Computer-Aided Design (ICCAD-91),* pp. 248-251, Nov. 1991.

[2] S. Barbagallo, F. Corno, P. Prinetto, and M. Sonza Reorda, "Testing a Switching Memory in a Telecommunication System," *Proc. IEEE Int'l Test Conf.* pp. 947-953, Oct. 1995.

[3] S. Barbagallo, D. Medina, F. Corno, P. Prinetto, and M. Sonza Reorda, "Integrating Online and Offline Testing of a Switching Memory," *IEEE Design & Test of Computers,* vol. 15, no. 1, pp. 63-70, Jan.-Mar. 1998.

[4] P.H. Bardell, W.H. McAnney, and J. Savir, "Built-In Test for VLSI," *Pseudorandom Techniques,* New York: John Wiley & Sons, 1987.

[5] H. Cheung and S.K. Gupta, "A BIST Methodology for Comprehensive Testing of RAM with Reduced Heat Dissipation," *Proc. IEEE Int'l Test Conf.,* pp. 386-395, Oct. 1996.

[6] B. Cockburn and Y.-F.N. Sat, "Synthesized Transparent BIST for Detecting Scrambled Pattern-Sensitive Faults in RAMs," *Proc. IEEE Int'l Test Conf.,* pp. 23-32, Oct. 1995.

[7] D.A. Fuentes and B. Courtois, "Random Pattern Testing versus Deterministic Testing of RAMs," *IEEE Trans. Computers,* vol. 38, no. 5, pp. 637-650, May 1989.

[8] R. Dekker, F. Beenker, and L. Thijssen, "Realistic Built-In Self-Test for Static RAMs," *IEEE Design & Test of Computers,* vol. 6, no. 1, pp. 26-34, Feb. 1989.

[9] I.D.A. Dornier Science Data Store, Crouzet, Document No. LSDS-FR-1000-DS, pp. 95-97, 1998.

[10] A.J. Van de Goor, *Testing Semiconductor Memories, Theory and Practice.* Chichester: John Wiley & Sons, 1991.

[11] A.J. Van de Goor, "Using March Tests to Test SRAMs," *IEEE Design & Test of Computers,* vol. 10, no. 1, pp. 8-14, Mar. 1993.

[12] S. Hellebrand, H.-J. Wunderlich, A. Ivaniuk, Y. Klimets, and V.N. Yarmolik, "Error Detecting Refreshment for Embedded DRAMs," *Proc. 17th VLSI Test Symp.,* 1999.

[13] J.L. Hennessy and D.A. Patterson, *Computer Architecture—A Quantitative Approach.* San Mateo, Calif.: Morgan Kaufmann, 1990.

[14] O. Kebichi, M. Nicolaidis, and V.N. Yarmolik, "Exact Aliasing Computation for RAM BIST," *Proc. IEEE Int'l Test Conf.,* pp. 13-22, 1991.

[15] K. Kinoshita and K.K. Saluja, "Built-In Testing of Memory Using an On-Chip Compact Testing Scheme," *IEEE Trans. Computers,* vol. 35, no. 10, pp. 862-870, Oct. 1986.

[16] K.T. Le and K.K. Saluja, "A Novel Approach for Testing Memories Using a Built-In Self-Testing Technique," *Proc. IEEE Int'l Test Conf.,* pp. 830-839, 1996.

[17] *DRAM Data Book.* Micron Technology Inc., 1998.

[18] B. Nadeau-Dostie, A. Silburt, and V.K. Agarwal, "Serial Interfacing for Embedded-Memory Testing," *IEEE Design & Test of Computers,* vol. 7, no. 2, pp. 52-64, Apr. 1990.

[19] M. Nicolaidis, "Transparent BIST for RAMs," *Proc. IEEE Int'l Test Conf.,* pp. 598-607, Oct. 1992.

[20] P. Olivo and M. Dalpasso, "Self-Learning Signature Analysis for Non-Volatile Memory Testing," *Proc. IEEE Int'l Test Conf.,* pp. 303-308. Oct. 1996.

[21] I.M. Ratiu and H.B. Bakoglu, "Pseudo-Random Built-In Self-Test Methodology and Implementation for the IBM RISC System/6000 Processor," *IBM J. Research and Development,* vol. 34, no. 1, pp. 78-84, 1990.

[22] T.R.N. Rao and E. Fujiwara, *Error-Control Coding for Computer Systems.* Englewood Cliffs, N.J.: Prentice Hall Inc., 1989.

[23] N. Sakashita et al., "A Built-in Self-Test Circuit with Timing Margin Test Function in a 1Gbit Synchronous DRAM," *Proc. IEEE Int'l Test Conf.,* pp. 319-324, 1996.

[24] K.K. Saluja, R. Sharma, and C.R. Kime, "A Concurrent Testing Technique for Digital Circuits," *IEEE Trans. Computer-Aided Design of Circuits and Systems,* vol. 7, no. 12, pp. 1250-1260, Dec. 1988.

[25] N. Wehn and S. Hein, "Embedded DRAM Architectural Trade-Offs," *Proc. Design, Automation and Test in Europe Conf. (DATE 98),* pp. 704-708, 1998.

[26] V.N. Yarmolik, S. Hellebrand, and H.-J. Wunderlich, "Self-Adjusting Output Data Compression: An Efficient BIST Technique for RAMs," *Proc. Design, Automation and Test in Europe Conf. (DATE '98),* pp. 173-179, 1998.

[27] V.N. Yarmolik, "Analysis of Signature Testability of Digital Circuits," *Automation and Remote Control,* pp. 1437-1443, Mar. 1990.
[28] Y. You and J.P. Hayes, "A Self-Testing Dynamic RAM Chip," *IEEE JSSC,* pp. 428-435, Feb. 1985.