17th IEEE VLSI Test Symposium, Dana Point, California, April 25 - April 29, 1999

Error Detecting Refreshment for Embedded DRAMs

S. Hellebrand, H.-J. Wunderlich
Division of Computer Architecture
University of Stuttgart
Germany

A. Ivaniuk, Y. Klimets, V. N. Yarmolik
Computer Systems Department
Belarussian State University of Informatics and
Radioelectronics, Minsk, Belarus

Abstract

This paper presents a new technique for on-line consistency checking of embedded DRAMs. The basic idea is to use the periodic refresh operation for concurrently computing a test characteristic of the memory contents and compare it to a precomputed reference characteristic. Experiments show that the proposed technique significantly reduces the time between the occurrence of an error and its detection (error detection latency). It also achieves a very high error coverage at low hardware costs. Therefore it perfectly complements standard on-line checking approaches relying on error detecting codes, where the detection of certain types of errors is guaranteed, but only during READ operations accessing the erroneous data.

1 Introduction

Present day systems-on-a-chip typically integrate a variety of different components like processor cores, SRAMs, ROMs and user defined logic on a single chip. Growing integration densities have made it feasible to embed dynamic RAM cores of considerable sizes, too [22]. 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 - 8, 10, 12 - 14, 16, 17, 19, 21, 24]. With increasing memory densities the relative area for the BIST resources becomes negligible.

This work was supported by BMBF-Grant WEI-X023.2.

To deal with soft errors during system operation, adding standard on-line checking capabilities based on error detecting codes is the first step also for embedded DRAMs [20]. Depending on the type of 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, here errors should be detected as early as possible to allow for recovery before the data are requested by the system. In this case, periodic transparent tests or consistency checks during idle times of the memory offer a solution, but the intervals between test phases may still be too long [3, 17, 18, 23].

In this paper it will be shown that the necessary BIST equipment can be efficiently re-used for implementing an on-line test method which complements or partly replaces conventional on-line checking strategies, such that low error detection latencies are guaranteed. The basic idea of the proposed approach is to use the periodic refresh operation for concurrently computing a memory characteristic C_{TEST} and compare it to a precomputed reference characteristic C_{REF}. Conflicts between C_{TEST} and C_{REF} then indicate a soft error. In the following Section 2 the requirements such a characteristic has to meet will be explained in detail. In Section 3, useful concepts of earlier work will be briefly reviewed, and the proposed memory architecture with error detecting refreshment will be presented in Section 4. To evaluate the new architecture experiments have been performed relying both on random simulations and on the simulation of real program data. The results documented in Section 5 will show that the proposed approach in fact combines a high error detection rate with a low error detection latency.

2 Using the Periodic Refresh Operation for Consistency Checking

Recently, techniques for periodic consistency checking of embedded memories have been proposed which re-use the resources common to most BIST approaches for memories (see Figure 1) [18, 23].

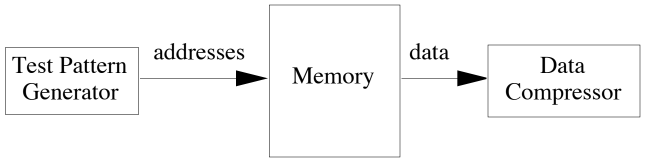


Figure 1: Typical BIST architecture for memories.

The test pattern generator produces a predetermined sequence of memory addresses, and the corresponding sequence of data is fed into the data compressor, the final state of which represents a characteristic of the memory contents. For the initial correct memory contents a reference characteristic C_{REF} is „learned" this way (or precomputed, if the memory contents is known after a reset) and saved in a specific location on chip. The same procedure is repeated periodically, and the respective characteristics C_{TEST} are compared to C_{REF} to reveal inconsistencies. These techniques were proposed as off-line BIST techniques to be applied during idle times of the memory. In this section we will discuss the ideal features of a technique for consistency checking to be used on-line during memory operation.

To overcome the problem of data retention dynamic RAMs refresh data during READ/WRITE operations and during periodic refresh operations. In a typical memory organization as shown in Figure 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. 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 the test characteristic C_{TEST} introduced above. However, in contrast to an off-line 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. As a consequence the characteristic has to meet the following two requirements:

- There must be a simple and fast mechanism to update the reference characteristic concurrently with changes in the memory contents, such that the periodic checks can be performed without repeating the initial learning phase.
- It should be possible to build the characteristic iteratively from „row characteristics". The time to compute a row characteristic must not exceed the time to refresh a row.

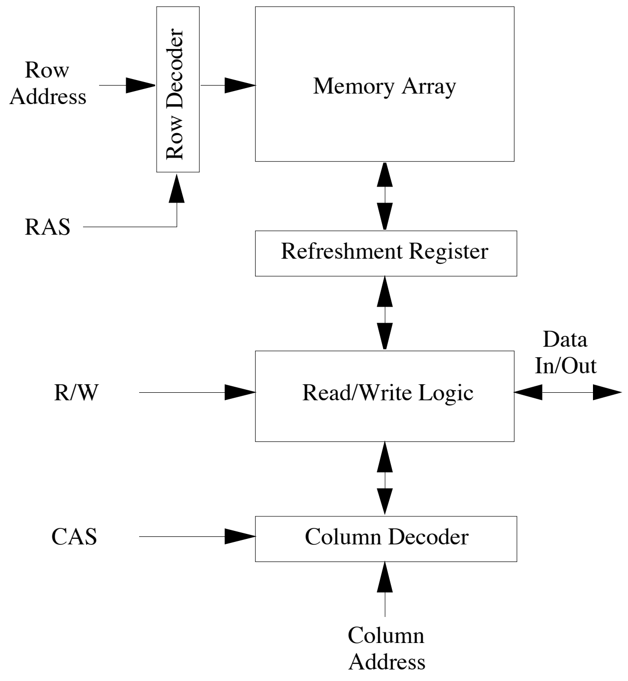


Figure 2: Typical organization of a dynamic RAM.

Consistency checking based on signature analysis as described in [18] does not fulfill these constraints, because changes in memory require to completely recompute C_{REF} before the checking can be started. Also, signature computation is done bit by bit (word by word), which implies that for each row n clock cycles are necessary, when n denotes the number of columns in the memory array. The „modulo-2 address characteristic" introduced in [23] is self-adjusting, i.e. after WRITE operations C_{REF} can be adjusted in one step. As shown in the next section, it can be implemented in such a way, that the second requirement is met, too.

3 Self-Adjusting Data Compression

In this section the basic principles and properties of the modulo-2 address characteristic are briefly reviewed. In particular, it will be demonstrated that the corresponding data compressor can be generalized in such a way, that the partial characteristic associated with each row in the memory array can be computed in one clock cycle.

As shown in Figure 3, the modulo-2 address characteristic compresses the memory contents to a characteristic C obtained as the bitwise modulo-2 sum of all addresses pointing to „1“. It allows to implement periodic 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

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

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

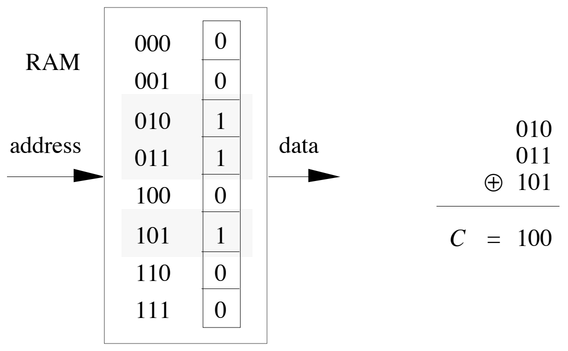


Figure 3: Modulo-2 address characteristic for bit-oriented RAMs.

For computing the complete characteristic, as well as for updating it concurrently with WRITE operations, the simple compressor circuit of Figure 4 can be used which performs bitwise EXOR operations on the address lines controlled by the data input.

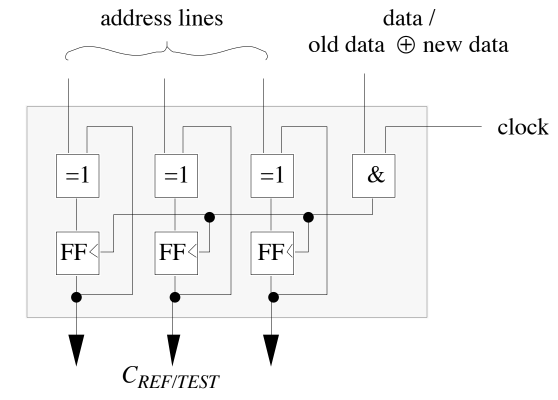


Figure 4: Data compressor based on the modulo-2 address characteristic.

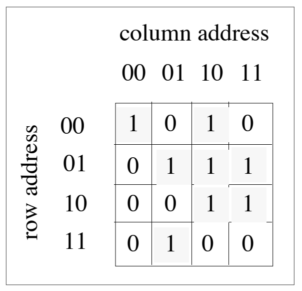
To calculate the initial characteristic C_{REF} or the test characteristic C_{TEST} a counter or an LFSR has to generate all memory addresses. If the memory operation starts after a reset to zero, C_{REF} is known to be zero, and the initialization can be skipped.

In [23] it has been shown that the modulo-2 address characteristic provides the same quality as characterizing the memory by conventional signature analysis:

1) All single errors are detectable and diagnosable. The expression $C_{REF} \oplus C_{TEST}$ provides the address of the faulty memory cell in this case.
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.

In contrast to conventional signature analysis, however, changes in memory do not require the time-consuming recomputation of C_{REF}. Adjusting the characteristic requires to compare the old and the new contents, and an efficient implementation strongly depends on the memory organization. In the case of dynamic RAMs the old memory contents is usually transferred to the refreshment register before it is overwritten. Therefore the difference $M[a^*]^{new} \oplus M[a^*]^{old}$ can be calculated without extra READ operations or performance losses [23].

However, the basic technique described so far is not efficient enough to be applied during a periodic refresh operation, because it steps through the memory bit by bit. The data compressor of Figure 4 must be generalized to compute the partial characteristic corresponding to one row in one step (see Figure 5).



row characteristics

$$\begin{array}{lclcl}
C_{00} & = & 0000 \oplus 0010 & = & 0010 \\
C_{01} & = & 0101 \oplus 0110 \oplus 0111 & = & 0100 \\
C_{10} & = & 1010 \oplus 1011 & = & 0001 \\
C_{11} & & & = & 1101 \\
\hline
C_{REF/TEST} & & & = & 1010
\end{array}$$

Figure 5: Row-wise computation of the modulo-2 address characteristic.

If memory addresses are split into row and column addresses $a = (a_r, a_c)$ and $A_1(r) := \{a_c \mid M[a_r, a_c] = 1\}$ 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 iteratively obtained 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 Figure 6.

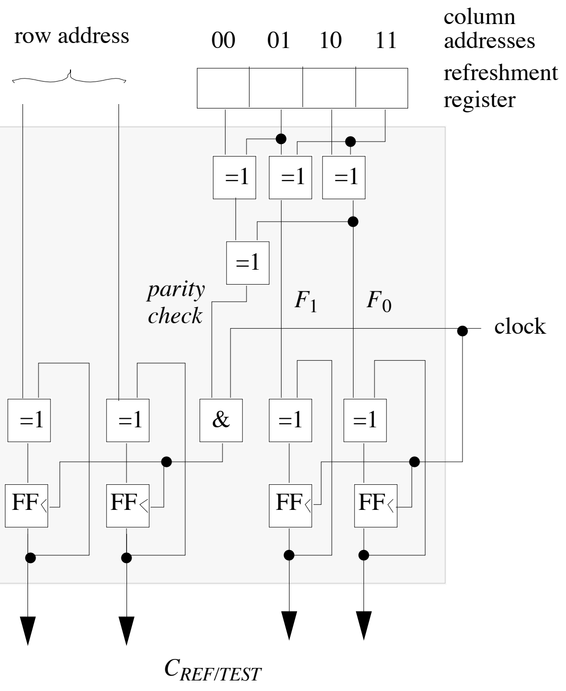


Figure 6: Data compressor for the fast computation of row characteristics.

The row 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 bitwise 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 denoting the i-th component of a_c, $0 \le i < l$. As only bit-addresses with $a_c = 1$ can contribute to the i-th sum, it is sufficient to implement functions F_i which count (modulo 2) the number of ones at all the addresses with this property. The second component of C_r is then derived as

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

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

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

EXOR-gates. Overall, the output data compressor of Figure 6 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. With l - 2 levels in the EXOR tree for the parity check and the functions $F_0, \ldots, F_{l-1}$, the depth of this network is l - 1.

Assuming a 1024 × 1024 bit dynamic memory, the data compressor can for example be implemented with 20 flip-flops, 2056 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 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 [15].

4 The Complete Memory Architecture

This section briefly sketches the complete architecture of an embedded DRAM with error detecting refreshment. Its core is the generalized data compressor described in Section 3, which can of course also be used to speed up the computation of the initial characteristic C_{REF} (if necessary) and to adjust C_{REF} during normal memory operation. As explained in Section 3, 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

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

provides a very simple and efficient architecture for DRAMs with error detecting refreshment (see Figure 7).

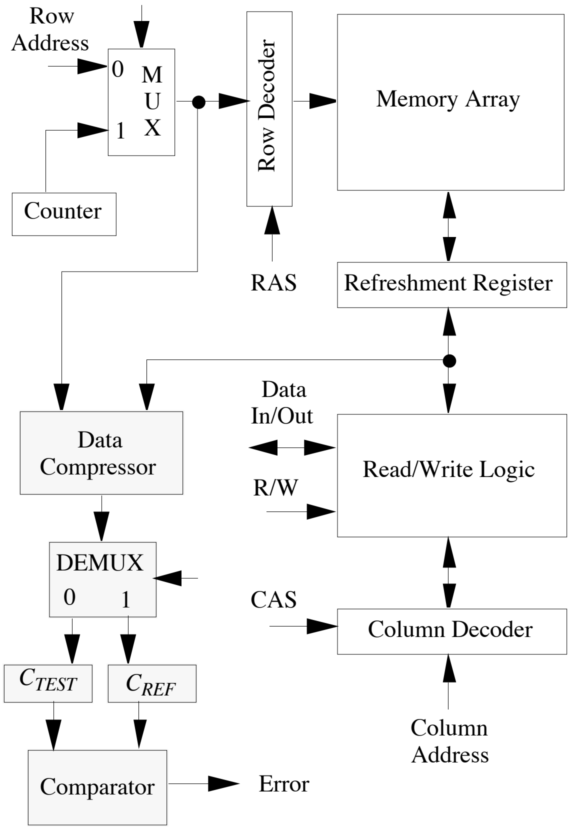


Figure 7: Complete architecture for a DRAM with error detecting refreshment.

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 initiate a concurrent update of C_{REF}. In the first phase the memory row with its old contents is 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 $_{CTEST}$ 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 Figure 7). With the figures given above, this is negligible compared to the overall area of the memory. Consequently, the cost for an embedded DRAM with error detecting refreshment is much lower than the cost for implementing error detecting codes. The expected fault coverage is similar, but obtained with a reduced latency.

5 Experimental Evaluation

To evaluate the proposed scheme it was compared to a standard on-line 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, even for simple parity checking, reading an erroneous cell is equivalent to detecting single errors. 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 [15]:

- 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 production 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 Figure 8 and Figure 9.

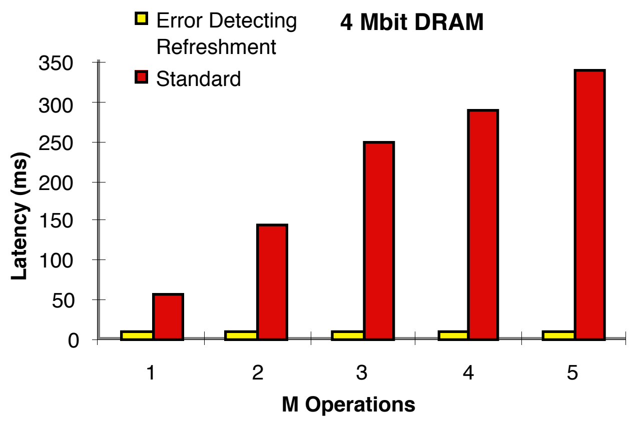


Figure 8: Average error detection latency for a 4 Mbit DRAM.

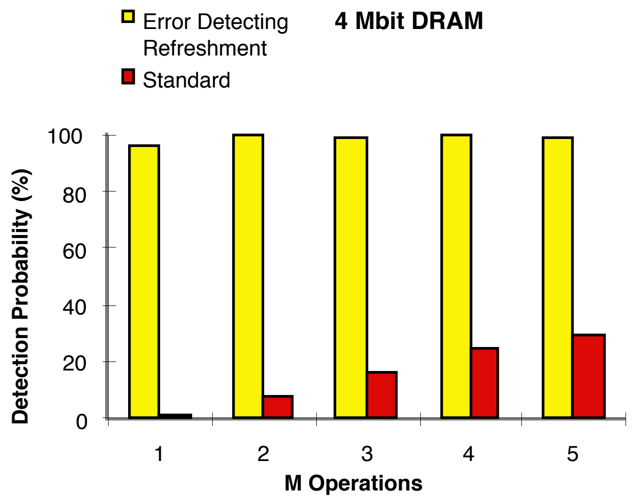


Figure 9: Average detection probability for a 4 Mbit DRAM.

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

- The average error detection latency for the proposed technique is around 8 ms in all experiments, which is approximately 50% of a refresh period.
- The error detection latency for the standard approach 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% in all experiments. Only in rare cases errors were 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%. In the worst case, it is even below 10%. However, it should be noted that here errors remained undetected, only because the corresponding data were not requested by the random sequence, and that all test sequences still provided the correct results.

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 [11]. The mechanism for fault injection was the same as for the random simulations. The results with respect to error detection latency and fault coverage are presented in Table 1 and Table 2.

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 1: Error detection latency for the benchmark programs SPICE, TeX, and GCC.

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

Table 2: Error coverage for the benchmark programs SPICE, TeX, and GCC.

For error detecting refreshment similar results are observed as described above. For the standard approach, however, considerably higher latencies and lower fault coverages are obtained. This is due to the reduced number of READ operations in the benchmark programs.

These results clearly show that error detecting refreshment complements standard on-line checking in an ideal way. On average, the detection latency is very low, and only a few errors escape. For those errors on-line checking still guarantees detection, although only during READ operations.

6 Conclusions

A new technique for on-line 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 periodic refresh operations. 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 even partly replace conventional on-line checking schemes based on error detecting codes, where the detection of certain types of errors is guaranteed, but high detection latencies must be expected.

7 References

1 V. C. Alves, M. Nicolaidis, P. Lestrat, and B. Courtois: Built-in Self-Test for Multi-Port RAMs; Proc. IEEE Int. Conf. on Computer-Aided Design, ICCAD-91, November 1991, pp. 248-251.

2 S. Barbagallo, F. Corno, P. Prinetto, M. Sonza Reorda: Testing a Switching Memory in a Telecommunication System; Proc. IEEE Int. Test Conf., Washington, DC, Oct. 1995, pp. 947-953.

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, January-March 1998, pp. 63-70.

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, S. K. Gupta: A BIST Methodology for Comprehensive Testing of RAM with Reduced Heat Dissipation; Proc. IEEE Int. Test Conf., Washington, DC, Oct. 1996, pp. 386-395.

6 B. Cockburn, Y.-F. N. Sat: Synthesized Transparent BIST for Detecting Scrambled Pattern-Sensitive Faults in RAMs; Proc. IEEE Int. Test Conf., Washington, DC, Oct. 1995, pp. 23-32.

7 R. David, A. Fuentes, and B. Courtois: Random Pattern Testing Versus Deterministic Testing of RAMs; IEEE Trans. on Comp., Vol. C-38, No. 5, May 1989, pp. 637-650

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, Feb. 1989, pp.26-34.

9 Dornier, IDA, Science Data Store, Crouzet; Document No. LSDS-FR-1000-DS, 1988, pp. 95-97.

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

11 J. L. Hennessy, D. A. Patterson: Computer Architecture – A Quantitative Approach; San Mateo, California: Morgan Kaufmann Publishers, Inc., 1990.

12 O. Kebichi, M. Nicolaidis, V.N.Yarmolik: Exact Aliasing Computation for RAM BIST; Proc. IEEE Int. Test Conf., Washington, DC, Oct. 1995, pp. 13-22.

13 K. Kinoshita, K. K. Saluja: Built-In Testing of Memory Using an On-Chip Compact Testing Scheme; IEEE Trans. on Comp., Vol. C-35, No. 10, October 1986, pp. 862-870.

14 K. T. Le, K. K. Saluja: A Novel Approach for Testing Memories Using a Built-In Self-Testing Technique, Proc. IEEE Int. Test Conf., Washington, DC, 1986, pp. 830-839.

15 DRAM data book. Micron Technolgy Inc., 1998.

16 B. Nadeau-Dostie, A. Silburt, and V. K. Agarwal: Serial Interfacing for Embedded-Memory Testing; IEEE Design & Test of Computers, Vol. 7, No. 2, April 1990, pp. 52-64.

17 M. Nicolaidis: Transparent BIST for RAMs; Proc. IEEE Int. Test Conf., Baltimore, MD, Oct. 1992, pp. 598-607.

18 P. Olivo, M. Dalpasso: Self-Learning Signature Analysis for Non-Volatile Memory Testing; Proc. IEEE Int. Test Conf., Washington, DC, Oct. 1996, pp. 303-308.

19 I. M. Ratiu, H. B. Bakoglu: Pseudo-random built-in self-test methodology and implementation for the IBM RISC System/6000 processor; IBM Journal of Research and Development, Vol. 34, No. 1, 1990, pp. 78-84

20 T. R. N. Rao, E. Fujiwara: Error-Control Coding for Computer Systems; Englewood Cliffs, NJ: Prenctice Hall, Inc., 1989.

21 N. Sakashita et al.: A Built-in Self-Test Circuit with Timing Margin Test Function in a 1Gbit Synchronous DRAM; Proc. IEEE Int. Test Conf., Washington, DC, Oct. 1996, pp. 319-324.

22 N. Wehn, S. Hein: Embedded DRAM Architectural Trade-Offs; Proc. Design, Automation and Test in Europe Conf. 1998 (DATE 98), Paris, 1998, pp. 704-708.

23 V. N. Yarmolik, S. Hellebrand, H.-J. Wunderlich: Self-Adjusting Output Data Compression: An Efficient BIST Technique for RAMs; Proc. Design, Automation and Test in Europe 1998 (DATE 98), Paris, 1998, pp. 173-179

24 Y. You, J. P. Hayes: A self-testing dynamic RAM chip; IEEE Journal of Solid-State Circuits, February 1985, pp. 428-435.