An On-line BIST RAM Architecture with Self Repair Capabilities

An On-Line BIST RAM Architecture With Self-Repair Capabilities

Alfredo Benso, Silvia Chiusano, Giorgio Di Natale, and Paolo Prinetto, *Member, IEEE*

***Abstract*—The emerging field of Self-Repair Computing is expected to have a major impact on deployable systems for space missions and defense applications, where high reliability, availability, and serviceability are needed. In this context, RAM (random access memories) are among the most critical components. This paper proposes a built-in self-repair (BISR) approach for RAM cores. The proposed design, introducing minimal and technology-dependent overheads, can detect and repair a wide range of memory faults including: stuck-at, coupling, and address faults. The test and repair capabilities are used on-line, and are completely transparent to the external user, who can use the memory without any change in the memory-access protocol. Using a fault-injection environment that can emulate the occurrence of faults inside the module, the effectiveness of the proposed architecture in terms of both fault detection and repairing capability was verified. Memories of various sizes have been considered to evaluate the area-overhead introduced by this proposed architecture.**

***Index Terms*—Built-in self-repair, built-in self-test, on-line testing.**

ACRONYMS[1]

BIRA	built-in redundancy-allocation
BISD	built-in self-diagnosis
BISR	built-in self-repair
BIST	built-in self-test
BISTAR	built-in self-test and repair
CAM	content addressable memory
CF	coupling fault
EOP	end of production
FPGA	field programmable gate array
RAM	random access memory
RISC	reduced instruction set computer
SRAM	static RAM.

NOTATION

c_x, c_j	RAM cells
K	number of spare cells in the RAM
l_i	a line of the CAM
m	RAM data width
n_j	a neighborhood cell of c_x $(0 \leq j \leq 7)$
N	nominal addressing space
s_i	a spare cell in the RAM.

Manuscript received November 11, 1999; revised June 16, 2000. This work was supported in part by the Istituto Superiore Mario Boella under the project Test D.O.C.: Quality and Reliability of Complex System-on-Chip.

The authors are with the Dipartimento di Automatica e Informatica, Politecnico di Torino, I-10129, Torino, Italy (e-mail: {Alfredo.Benso; Silvia.Chiusano; DiNatale; Paolo.Prinetto}@polito.it).

Publisher Item Identifier S 0018-9529(02)02955-X.

[1]The singular and plural of an acronym are always spelled the same.

I. INTRODUCTION

THE EMERGING field of self-repair computing is expected to have a major impact on deployable systems for space missions and defense applications that need to survive and perform at optimal functionality during long duration in unknown, harsh and/or changing environments. Examples of such applications include outer solar system exploration, missions to comets and planets with severe environmental conditions, long lasting space-borne surveillance platforms, defffensive counter-measures, long-term nuclear waste, and other hazardous environment monitoring and control. Self-repair computing is also expected to greatly enrich the area of commercial applications in which high availability and serviceability are needed; such applications range from biomedical devices to automotive applications.

The process of repairing a RAM can be divided into several steps. In a first phase, a test algorithm is executed on the memory array. If a fault is detected, it is necessary to locate it (diagnosis) and to allocate redundant memory space to replace the faulty cell. When these operations are built-in the RAM architecture, the steps are named: BIST, BISD, BIRA, BISR.

There are 3 possible solutions to insert redundant space into a memory array:

1) Row/Column Only: The memory contains spare rows or spare columns. When a fault must be repaired, the row/column containing the fault is replaced with one of the spare row/columns. This solution allows the manufacturer to repair faulty cells easily, but it does not allow optimal use of redundant space because repairing a single fault requires allocating a whole spare row or column.
2) Row–Column: The memory contains both spare rows and spare columns. Each fault can be repaired, by using either a spare row or a spare column. When multiple faults are detected, this technique allows for more efficiently repairing the faults by selecting the best combination of spare rows/columns.
3) Cell-Only: Instead of repairing an entire row or column, when a fault is detected, only the address of the faulty cell is re-mapped on to a new cell; thus allowing an optimal allocation of the redundant space.

This paper proposes an innovative architecture for SRAM, characterized by BISR capabilities based on cell-only redundant space allocation at the user level. The memory is not electrically

repaired, but spare cells replace faulty ones, using an on-line address re-mapping scheme. The repair process is transparent to the user, and is independent from the memory physical-implementation.

In this approach, the self-repair architecture is coupled with an *ad-hoc* defined on-line transparent BIST algorithm. The on-line BIST is therefore executed concurrently with the memory normal behavior, and is able to detect the appearance of a wide range of faults, including coupling faults, usually not detectable during end-of-production or power-up tests. These faults have a higher probability of appearing in very high-density RAM only when the circuit reaches high temperatures. Because this condition can not be guaranteed except after a long period of use, EOP or power-up tests usually do not provide a good coverage of coupling faults.

The on-line BIST algorithm also allows implementing a very efficient BISD and redundancy-allocation strategy.

To assess the quality of the proposed architecture, a simulation-based fault injection environment has been created to emulate the appearance of various faults in the memory module. Experiments were performed to validate the detection capability of the BIST circuitry first, and then the functionality of the BISR logic.

Section II briefly presents some related research in self-repairing computing. Section III presents the conceptual scheme of the proposed architecture, focusing on the on-line BIST algorithm and the self-repair architecture. Section IV discusses some issues raised during the implementation of the approach. Section V presents the fault-injection environment set up to validate the BIST and BISR capabilities. Section VI discusses experimental results concerning the area overhead and the BISR logic fault coverage. Section VII summarizes the most interesting results of the proposed approach.

II. State of the Art

Most of the research activities on self-repair techniques were focused on FPGA [1]–[5]. BIST and BISR schemes have been proposed as potential solutions to the problem of repairing memories, mainly at the manufacturer level (at the EOP) [6]–[12]. The scheme in [6] limits self-repair to field-failures only. To remove manufacturing defects, it uses the traditional row-column repair approach, based on an on-chip micro-programmed BIST scheme and self-repair logic block, with a spare memory block. Reference [7] uses an elaborate on-chip RISC processor to collect and analyze full failure bitmaps to figure out a repair solution. Besides the complexity of the RISC processor, the method also requires that a large enough fault-free block of the RAM under test must be available to store the failure bitmap. Reference [8] considers ultra-large capacity single-chip memories. The proposed architecture uses a hierarchical organization to achieve optimal conditions for memory access time. References [9]–[12] analyze algorithms that optimize the repair solution for a given bit-failure pattern in a redundant RAM.

All these solutions are typically adopted for stand-alone memories, where it is possible (at the end of production) to do hardware repairing through anti-fuse/laser techniques.

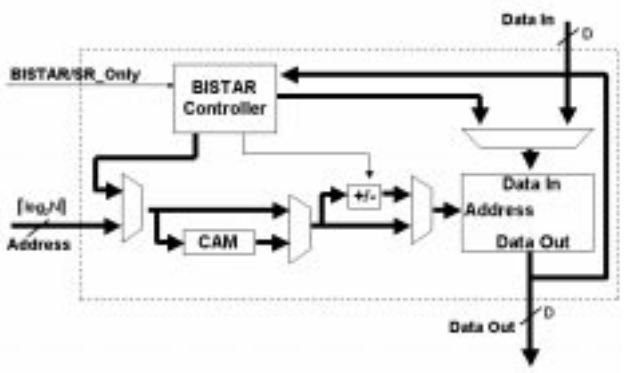

Fig. 1. BISTAR conceptual architecture.

Nevertheless, in today's high integrated circuits that embed large memories as well as other heterogeneous types of cores (digital logic, analog block), the very low, if not null, physical accessibility to the cores makes hardware repairing not feasible or cost effective.

Recently, new techniques have been introduced to perform a memory repair through self-reconfiguration of the addressing space [13]–[15]. In [13], [14], BIST and BISR architectures are inserted into the RAM. The self-test begins at the system power-up and the information on the faulty cells is stored and used thereinafter to reconfigure the memory. In the self-repair circuit, two registers are inserted for each redundant row or column. The first register stores the address of a faulty row (or column) whereas the second register stores the address of the row (or column) that replaces the faulty one. When a faulty cell is addressed from the external, the circuit reacts by changing the address value to the correct one. This solution is very expensive in term of routing overhead. For each redundant row or column, a set of signals (for addressing the memory) is routed from the BISR circuits to the memory. Reference [14] considers a column-only repair strategy to simplify the spare allocation procedure.

III. The Conceptual Architecture

The conceptual idea underlying the proposed approach is to couple an on-line transparent BIST algorithm with a "functional self-repair" architecture in the same BISTAR logic.

"Functional self-repair" means that a faulty cell must be replaced by a spare one using an address re-mapping scheme. The BIST part of the logic executes an on-line test, based on a linear algorithm, to detect single stuck-at, transition, coupling, and address faults.

Fig. 1 is a conceptual view of the BISTAR architecture. Section IV presents the actual implementation of the BISTAR logic aiming at minimizing critical paths. The self-repair logic is based on a CAM used to re-map the address of the faulty cells. The BISTAR controller is in charge of executing the test algorithm and controlling the repair procedures.

To make the approach more general, and to allow the user to fulfill power budget constraints and to perform diagnosis from the outside, the core has two possible functional modes, selected via the input signal `BISTAR/SR_only`:

1) `SR_only`: the BIST algorithm is disabled, but the self-repair capability is active. Despite new faults not being detected, the re-mapping addressing mechanism is still active for previously detected faulty cells.
2) `BISTAR`: both self-test and self-repair capabilities are enabled. The BISTAR controller continuously executes the test algorithm and possibly repairs faulty cells.

The self-repair capability is exploited during the self-test of the memory array in order to guarantee the transparency from the user point of view. The proposed approach mainly includes three phases executed sequentially: isolation, test execution, and repairing or restoring. Each c_x under test is isolated by functionally replacing it with one of the available s_i: the content of c_x is copied into s_i and its address is stored into the CAM. During the test, any external operation on c_x is actually performed on s_i. The c_x is then tested on-line by executing the algorithm in Section III-B. Not to degrade the memory performance, the test execution is temporary suspended to serve any external memory access request that could occur during the test itself. At the test completion, if no fault has been detected, the content of c_x is restored and its address erased from the CAM; otherwise, s_i is thereinafter used as a repair cell for c_x.

A. Memory Built-In Self-Repairing

The proposed BISR strategy aims at keeping constant the memory storing-capability seen by the user. Faulty cells are functionally replaced by spare ones via a dynamic on-the-fly reconfiguration of the memory-addressing space.

From an external user point of view, the memory has a nominal addressing space of N cells, of m-bits each. The actual memory module, instead, has an effective storage capacitance of $N + K$ cells.

To optimize the allocation of the redundant memory space, the approach is based on a cell-only repair strategy: when a fault is detected in a cell, instead of repairing an entire row or column, only the faulty cell is re-mapped on a spare one.

Address re-mapping is achieved by a K-lines CAM, each l_i corresponding to an s_i. In particular, the l_i, $0 \leq i < K$, of the CAM stores the address of a faulty cell c_j, $0 \leq j < N + K$. In this way c_j is functionally replaced by the s_i. This solution allows reducing the area and the routing overhead. Instead of using an additional register into the CAM in which is stored the address of the redundant cell [13], [14], the association between the line position and the redundant cell is hardwired. The repairable memory-array space includes all $N + K$ cells of the memory, thus allowing repair of spare cells as well.

Whenever a c_x of the memory is accessed, its address is first looked up in the CAM. Two cases can occur:

- 1) If c_x has been previously detected faulty (or is currently a cell under test), its address has been stored in the CAM. Then, when accessed, the CAM reacts with a hit, and outputs the address of the replace s_i, and a proper multiplexer routes it to the memory array. Any operation on the faulty c_x is thus performed on its replace s_i.
- 2) If a spare cell does not currently replace the target c_x, its address, not being stored in the CAM, is directly transferred to the memory array.

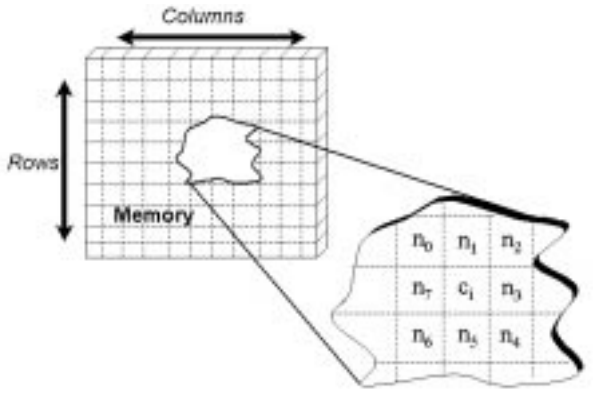

Fig. 2. Memory layout.

B. Memory Built-In Self-Testing

The proposed on-line self-test logic implements a custom transparent memory-test algorithm, which does not therefore affect the memory content.

To achieve high dependability, the memory must be repaired guaranteeing very low fault-latency. Moreover, an important variety of faults need to be targeted. A custom test algorithm has thus been adopted, optimized to exploit the knowledge of the memory layout. The algorithm has linear complexity and addresses faults both occurring inside a memory cell and involving pairs of physically adjacent cells. The assumption of knowing the memory layout does not limit the applicability of the method, because foundries usually provide details about the internal memory structure, and tools are available to extract this information (e.g., FlexStream™ by LSI Logic™ [16]). If this were not feasible, it is always possible to implement a quadratic algorithm targeting the same faults without requiring the knowledge of the memory internal structure.

Fig. 2 shows that the memory is tiled by a group of nonoverlapping neighborhoods. As far as the cardinality of the neighborhood set is concerned, the type-2 neighborhood choice of [17] has been used.

The implemented test algorithm targets the following faults:

- Stuck-at faults on the base cell;
- Transition faults on the base cell;
- Intra-word CF on the base cell for word oriented RAM;
- Inter-word CF between the base cell and its eight neighborhoods.

In particular, for both intra-word CF and inter-word CF, idempotent CF (CF_{id}) and inversion CF (CF_{in}) are covered [17].

The intra-word CF is detected by resorting to the "Walking 1/0" intermixed complement Data Background Sequence [18].

To detect inter-word CF, the following test is executed on each pair of adjacent cells c_x, n_j:

$$\{w_D(c_x),\, w_D(n_j),\, r_D(c_x),\, w_{\overline{D}}(n_j),\, r_D(c_x),\, w_D(n_j)$$
$$r_D(c_x),\, w_{\overline{D}}(c_x),\, w_D(n_j),\, r_{\overline{D}}(c_x),\, w_{\overline{D}}(n_j),\, r_{\overline{D}}(c_x)$$
$$w_D(n_j),\, r_{\overline{D}}(c_x)\}$$

- r represents read operations;
- w represents write operations;

- D represents any background pattern;
- $\overline{D}$ represents complement of D.

The r/w operation, which reads/writes on the base cell both a value D and its $\overline{D}$, covers stuck-at faults, whereas the transition fault on the base cell c_x are covered by the sequences:

$$\{w_D(c_x), r/w(n_1), r_D(c_x), r/w(n_1), w_{\overline{D}}(c_x), r/w(n_1), r_{\overline{D}}(c_x)\}, \{w_D(c_x), r/w(n_2), r_D(c_x)\}.$$

To minimize the test time, intra-word and inter-word testing are properly interleaved. For any pair of adjacent cells, c_x, n_j, a different background pattern is used. Whenever the number of needed patterns and the number of neighborhoods differ, the background patterns or the test of neighborhood pairs are repeated accordingly, to fill the gap.

The adopted algorithm does not specifically target "address decoder faults"; nevertheless, the following "address faults" are covered:

- c_x is not addressable because there is no link between the address decoder and the enable signal of the cell;
- c_x is not addressable, and address(c_x) accesses n_j;
- c_x is not addressed by address(c_x), and both n_j and c_x are reached by address(n_j);
- c_x is addressed by address(c_x), but both n_j and c_x are reached by address(n_j).

According to the FlexStream tool, a memory is considered organized by columns: cells that are adjacent inside a column have consecutive address into the memory. The functional address of the neighborhoods n_j can be easily computed based on the address of c_x:

$$\{\text{address}(n_j) | 0 \le j \le 7\} = \begin{cases} \text{address}(c_x) \pm \#\text{Rows} \pm 1, & j \in \{0, 2, 4, 6\} \\ \text{address}(c_x) \pm 1, & j \in \{1, 5\} \\ \text{address}(c_x) \pm \#\text{Rows} \pm 1, & j \in \{3, 7\}. \end{cases}$$

To reduce the complexity of the BIST controller, the memory is conceptually considered as a toroid, thus assuming:

- the left-hand-most and right-hand-most columns are adjacent;
- the top and bottom rows are adjacent.

This way, the test length is slightly increased (coupling faults with a very low occurrence-probability are tested), but the controller size is appreciably reduced.

The proposed algorithm has a complexity of $(8 \cdot 14)N = 112N$; $N \equiv$ the number of memory cells, $14 \equiv$ the number of memory accesses performed on each pair of neighborhood cells.

During testing, the cell-under-test, c_x and one of its neighborhood cells, n_j are isolated by replacing them with two spare cells: their original content is copied into the spare cells and the CAM content updated for address re-mapping. The test algorithm is then executed on the pair c_x, n_j. If no faults are detected, the original content of n_j is restored and its re-mapping address in the CAM removed. Then, the next neighborhood cell is considered. If the test is successful for all 8 pairs of c_x, n_j, then c_x is restored and the next cell of the memory is put under test. Otherwise, if c_x is faulty, then the test is repeated on the same pair of cells to distinguish between permanent and transient faults. If the repeated test fails as well, then c_x is considered as a permanent faulty cell and its functional replacement by the spare cell is not removed. Thus c_x is no longer accessible, and the memory is functionally repaired.

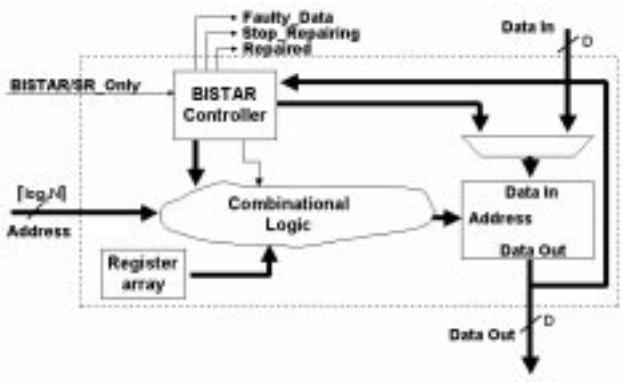

Fig. 3. Minimization of the address critical-path.

IV. Actual Implementation

To minimize the address critical-path of Fig. 1, the CAM structure has been implemented using a register array and a proper encoding logic, thus allowing the implementation of Fig. 3. There is no *a priori* constraint about the type of CAM one can use. Using a nonvolatile memory would keep the addressing-space reconfiguration status when the system is powered-down. The only performance degradation introduced with respect to the original memory protocol is a constant increase of the set-up time of the memory corresponding to the time required to propagate the address.

Three output signals (Faulty_data, Stop_repairing, and Repaired) are provided to increase further the dependability properties of the core:

- Faulty_data shows that the data read by the user are potentially corrupted. After a permanent repair, the signal is asserted whenever the user reads the content of a spare cell without having written it previously. In such a case, the content of the cell can be faulty. The signal is reset by the first writing operation on the replacing cell.
- Stop_repairing is asserted whenever no additional spare cells are available to execute the test. Thereinafter the SR_only functional mode (see Section II) is entered.
- Repaired shows that the addressed cell has been previously found faulty and replaced by a spare one. The signal can be helpful, e.g., for diagnosis purposes. To diagnose the memory, the user can:
  - force the module to enter the SR_only mode;
  - perform a read operation on any cell of the module;
  - for each address, check whether the Repaired is asserted or not. If asserted, the target cell is faulty.

V. BISTAR Validation

To validate the proposed BISTAR architecture, it is necessary to demonstrate that faults appearing in the memory are detected

TABLE I
FRACTION OF AREA-OVERHEAD INTRODUCED BY THE BISTAR ARCHITECTURE

Word Dimension	Number of Words	Number of Spare Cells 16	32	64
8-bits	1K	36.52%	68.45%	134.26%
	2K	22.81%	42.82%	83.98%
	4K	13.28%	24.98%	48.99%
	8K	7.47%	13.94%	27.34%
16-bits	1K	22.47%	41.08%	79.48%
	2K	14.25%	26.14%	50.62%
	4K	8.38%	15.43%	29.90%
	8K	4.86%	8.89%	17.24%
32-bits	1K	12.85%	22.74%	43.06%
	2K	8.39%	14.93%	28.32%
	4K	4.66%	8.32%	15.82%
	8K	2.62%	4.66%	8.87%

by the BIST circuitry, and that the BISR logic adequately reconfigures the memory address space. There are three steps:

1) One or more faults are injected into both the memory cells and addressing logic;
2) An interval of time is waited, until the BISTAR architecture localizes and repairs the fault;
3) The memory is exercised to verify its correct behavior after the repairing process.

To emulate a faulty memory, an *ad hoc* memory wrapper has been designed that can intercept data and address flows coming in and out of the memory, and then modify them according to a predefined fault model. The proposed approach is extremely flexible: the wrapper functionality is customizable to each specific experiment, defining, at simulation time, the type and number of faults to be injected into the memory.

The Fault Injector is described in VHDL as a set of modular blocks, each allowing the insertion of a particular fault. In performing the experiments, all the faults covered by the proposed test algorithm were injected.

To verify the correct memory behavior after each fault injection and the consequent repair process, a MATS$^+$ March Test algorithm [17] is performed.

VI. EXPERIMENTAL RESULTS

The experimental results presented in this section were gathered on several implementations of various-sized BISTAR architectures built around a static RAM m10p111hab, included in the LSI Logic™ G10 library [19]. In particular, to deeply-analyze the impact in terms of area overhead, all possible combinations of the following cores were synthesized by Synopsys™ [20] using the G10 library:

- 8, 16, and 32 bits word;
- 1k, 2k, 4k, and 8k words;
- 16, 32, 64 spare cells.

A. Area Overhead

Table I shows the synthesis results concerning the area overhead, expressed as fraction of the area added to achieve BISR and BIST functionality with respect to the area of the original memory module. Spare cells are considered as part of the BISR logic and therefore contribute to the area overhead.

TABLE II
BISR LOGIC FAULT COVERAGE

MODULE	Total	Repaired
BISR Controller	972	92.7%
BISR Logic (CAM, *etc*)	2224	89.16%

To provide the designer with a quick, approximate estimation of the area overhead, consider a memory of N words, K spare cells, and D bits word. The fraction of area overhead introduced by the BISR and BIST circuitry is

$$\begin{aligned}&\text{AreaOverhead}\\ &\quad=\frac{\text{Area}_{\text{BISRandBIST}}}{\text{Area}_{\text{RAM}}}\\ &\quad\equiv\frac{\text{Area}_{\text{spare}}+\text{Area}_{\text{CAM}}+\text{Area}_{\text{BISTAR}}+\text{Area}_{\text{routing}}}{}\end{aligned}\tag{1}$$

$$\begin{cases}\text{Area}_{\text{RAM}}=O(N\cdot D)\\ \text{Area}_{\text{spare}}=O(K\cdot D)\\ \text{Area}_{\text{CAM}}=O(K\cdot\log_2(N))\\ \text{Area}_{\text{BISTAR}}=O(\log_2(N)),\quad \text{for } N>D.\end{cases}\tag{2}$$

Considering (2), the (1) reduces to:

$$\text{AreaOverhead}=\left(\frac{D+c\cdot\log_2(N)}{D\cdot N}\right)\cdot K.\tag{3}$$

For a given memory, the term in parentheses (3) is a constant because, N and D are fixed. As anticipated, (3) states that the area overhead is proportional to K. The constant c in (3) strongly depends on the target synthesis library. When dealing with the LSI Logic G10 library, c has been experimentally proved to be about 22.

B. BISR Logic Fault Coverage

To evaluate the repair capabilities of the BISR logic, set up the following experiment with three steps:

1) Inject one fault in the BISR logic using the Sunrise tool;
2) Wait an interval time, to allow the BISR architecture to localize the fault and consequently reconfigure the memory;
3) Test the memory using a March test, to verify its correct behavior after the reconfiguration.

Although the RAM core has not been explicitly designed to cover faults located in the BISR logic, the module was able to repair

- 92.7% of the faults inserted in the BISR Controller, and
- 89.16% of the faults inserted in the remaining part of the BISR logic; see Table II.

REFERENCES

[1] W. Mangione-Smith and B. Hutchings, "Configurable computing: The road ahead," in *Reconfigurable Architectures Workshop*, 1997.
[2] J. Lach, W. Mangione-Smith, and M. Potkonjak, "Efficiently supporting fault-tolerance in FPGAs," in *ACM/SIGDA Sixth Int. Symp. Field-Programmable Gate Arrays*, 1998.
[3] M. J. Wirthlin and B. L. Hutchings, "A dynamic instruction set computer," in *Proc. IEEE Symp. FPGA's for Custom Computing Machines*, 1995.
[4] E. Tau, D. Chen, and I. Eslick *et al.*, "A first generation DPGA implementation," in *Proc. Third Canadian Workshop on Field-Programmable Devices*, 1995, pp. 138–143.

[5] R. Bittner and P. Athanas, "Wormhole run-time reconfiguration," in *ACM/SIGDA Int. Symp. Field Programmable Gate Arrays*, 1997.
[6] H. Koile *et al.*, "A 30nsec 64Mb DRAM with built-in self-test and repair function," in *Int. Solid State Circuits Conf.*, 1992, pp. 150–151.
[7] R. Trueuer and V. K. Agarwal, "Built-in self-diagnosis for repairable embedded RAMs," *IEEE Design and Test of Computers*, pp. 24–33, 1993.
[8] T. Chen and G. Sunada, "Design of a self-testing and self-repairing structure for highly hierarchical ultra large capacity memory chips," *IEEE Trans. VLSI Syst.*, vol. 1, pp. 88–97, June 1993.
[9] J. R. Day, "A fault-driven, comprehensive redundancy algorithm," *IEEE Design & Test of Computers*, pp. 35–44, June 1985.
[10] R. W. Haddad, A. T. Dahbura, and A. B. Sharma, "Increased throughput for the testing and repair of RAM's with redundancy," *IEEE Trans. Computers*, vol. 40, pp. 154–166, Feb. 1991.
[11] N. Hasan and C. L. Liu, "Minimum fault coverage in reconfigurable arrays," in *Digest of Papers*, vol. FTCS-18, June 1988, pp. 348–353.
[12] D. K. Bhavsar, "An algorithm for row-column self-repair of RAM's and its implementation in the alpha 21 264," in *IEEE Int. Test Conf.*, 1999, pp. 311–318.
[13] O. S. Bair *et al.*, "Method and apparatus for configurable build-in self-repairing of Asic memories design," US Patent 5 577 050, Nov. 19, 1996.
[14] A. Kablanian *et al.*, "Built-in self repair system for embedded memories," US Patent 5 764 878, Jun. 9, 1998.
[15] I. Kim, Y. Zorian, and G. Komoriya *et al.*, "Built-in self repair for embedded high density SRAM," in *IEEE Int. Test Conf.*, 1998, pp. 1112–1119.
[16] LSI Logic, *FlexStream Reference Manual*, 1999.
[17] A. J. Van de Goor, *Testing Semiconductor Memories: Theory and Practice*. New York: Wiley, 1991.
[18] A. J. van de Goor and I. B. S. Tlili, "March tests for word-oriented memories," *IEEE Design, Automation and Test in Europe*, pp. 501–508, 1998.
[19] [Online]. Available: http://www.lsil.com
[20] Synopsys Inc., *VHDL Compiler Reference Manual*, 1994.

Alfredo Benso was born in Torino, Italy, in 1970. He received the M.S. degree in 1995 in computer engineering, and the Ph.D. degree in 1998 from the Politecnico di Torino, Italy.

He is a research assistant at the same university, where his research interests include design-for-testability techniques, BIST for complex digital systems, and dependability analysis of computer-based systems.

Silvia Chiusano was born in Torino, Italy, in 1970. He received the M.S. degree in 1996 in computer engineering from the Politecnico di Torino, Italy, and the Ph.D. degree in 2000.

Her research interests include high-level testable synthesis, high-level testing, design-for-testability techniques, and built-in self-test.

Giorgio Di Natale was born in Torino, Italy, in 1975. He received the M.S. degree in 1999 in computer engineering from the Politecnico di Torino, Italy, and is pursuing the Ph.D. degree at the same university.

His research interests include design-for-testability techniques, built-in self-repair, and FPGA testing.

Paolo Prinetto was born in Gassino Torinese, Italy, in 1953. He received the M.S. degree in 1976 in electronic engineering from the Politecnico di Torino, Italy.

Since 1990 he has been Full Professor of Computer Engineering at the same university, and since 1998 has also been joint professor at the University of Illinois at Chicago. His research interests cover testing, testable synthesis, BIST, and dependability.

He is a Golden Core Member of the IEEE Computer Society and is currently the elected Chair'n of the Test Technology Technical Council of the IEEE Computer Society.