Online and Offline BIST in IP-Core Design

Alfredo Benso, Silvia Chiusano, Giorgio Di Natale, and Paolo Prinetto
Politecnico di Torino

Monica Lobetti Bodoni
Siemens Information and Communication Networks

This article presents an online and offline built-in self-test architecture implemented as an SRAM intellectual-property core for telecommunication applications. The architecture combines fault-latency reduction, code-based fault detection, and architecture-based fault avoidance to meet reliability constraints.

THE TELECOMMUNICATION systems market—including digital communication systems and real-time fragmented-data-structure applications such as asynchronous-transfer-mode (ATM) switches and video-related products—has grown significantly in the past few years. To compete in this market, industries must achieve high performance, robustness, availability, and reliability in their products, while keeping production costs as low as possible. To guarantee high performance and large data-processing capacities, system designers have integrated digital, analog, and radio-frequency devices. At the same time, strong reliability constraints have contributed to the evolution of highly reliable digital components.

Memories are key components in telecommunication systems, playing a crucial role in system availability and serviceability. Memory components come in a wide variety of sizes, technologies (such as static RAM, dynamic RAM, and Rambus), and packaging (intellectual-property [IP] cores, chips, and dedicated boards). Regardless of their implementation, however, memories are vulnerable to permanent and transient faults caused by environmental stress and interference. This vulnerability stems from memory components' small circuit elements, which are subject to damage or unintended state changes from small amounts of energy.

In telecommunication applications, strict fault-latency requirements impose continuous memory testing to minimize the time between fault occurrence and detection and avoid interruption or degradation of system performance. Designers usually reach this goal with coexisting on- and offline built-in self-test techniques. Online BIST continuously tests the system for transient faults during its normal behavior. Offline BIST aims at detecting faults left undiscovered by online test, during dedicated time frames when system behavior is suspended.

The most common online-testing techniques rely on hardware redundancy (duplication or triplication), time redundancy, information redundancy (error-detecting and error-correcting codes), and parametric testing (built-in current sensors).[1] Whereas offline BIST architectures have been widely discussed in the literature,[2] and many standard offline solutions have been defined and evaluated, defining a standard online BIST approach is impossible. The design of such an architecture must be based on an analysis of the trade-offs between fault coverage

0740-7475/01/$10.00 © 2001 IEEE

and hardware or information redundancy overhead, and between fault latency and the system access rate.[3]

Whatever the detection mechanism, once a fault is detected, the system can respond with various strategies, depending on the reliability constraints:

- *Built-in self-repair (BISR)*. The memory architecture functionally removes the fault's effect. This is the solution needed for systems that cannot be physically repaired—for example, in deep-space applications.[4]
- *Graceful degradation*. The system executes its target application with some type of degradation—for example, a decrease in overall memory size.
- *Isolation*. The system signals detection of a fault and isolates itself from the external logic.
- *Detection only*. The system signals detection of an error but continues working. The external environment must handle the faulty condition.

At Politecnico di Torino, we have developed an advanced on- and offline RAM BIST architecture and implemented it as an IP core for Italtel, an Italian telecommunication system manufacturer.[5] Our approach combines four strategies to meet strict fault detection, fault tolerance, overhead, and system performance constraints: offline test, code-based fault detection, architecture-based fault avoidance, and architecture-based fault-latency reduction. We did not provide a full BISR capability because, in this application, engineers can physically repair the memory in a short time. However, the architecture allows BISR of at least one faulty cell, at the cost of test efficiency.

Design constraints

The target memory, a single-port static RAM clocked at 10K × 80 bits, acts as the command memory of an ATM switch-unit chip manufactured in LSI Logic's G11 technology. In addition to the command memory, the chip (PSE40K) contains approximately 700K random-logic gates and a 2.4-Mbit RAM devoted to data storage.

The SRAM's functional specifications required a low average access rate (one access every 200 cycles), thus allowing long idle intervals that we could exploit for testing. The memory-access strategy depends on the required operation. A write operation to the memory can separately access four 20-bit subwords. A read operation always accesses the entire 80-bit data word in parallel. For this reason, the memory has a write-enable signal for each input bit.

The technology we adopted imposed no power dissipation constraints. Finally, the area introduced for testing had to be as small as possible; duplication of the entire memory was not allowed.

Reliability constraints

The fault detection constraints required detecting but not necessarily correcting both permanent and transient faults in the memory cells as well as the addressing logic. The targeted permanent faults[2] included

- *single and multiple stuck-at*, where one or more bits of the memory array are stuck at a logic value;
- *coupling* (both intra- and interword), where a memory cell value or a transition modifies another cell's content;
- n-*bit burst*, where the first and last bits in a sequence of *n* faulty or correct bits are faulty; and
- *address faults*, which are stuck-at and coupling faults on address lines.

The targeted transient faults included

- *single-event and double-event upsets*, where the value of one or two bits of the memory array is flipped; and
- *address transient faults*, which are single-event and double-event upsets on address lines.

The fault tolerance constraints impose a short fault latency; faults must be detected within 100,000 cycles from their appearance (that is, within 500 memory accesses, considering the average access rate). Moreover, service should never be interrupted or degraded.

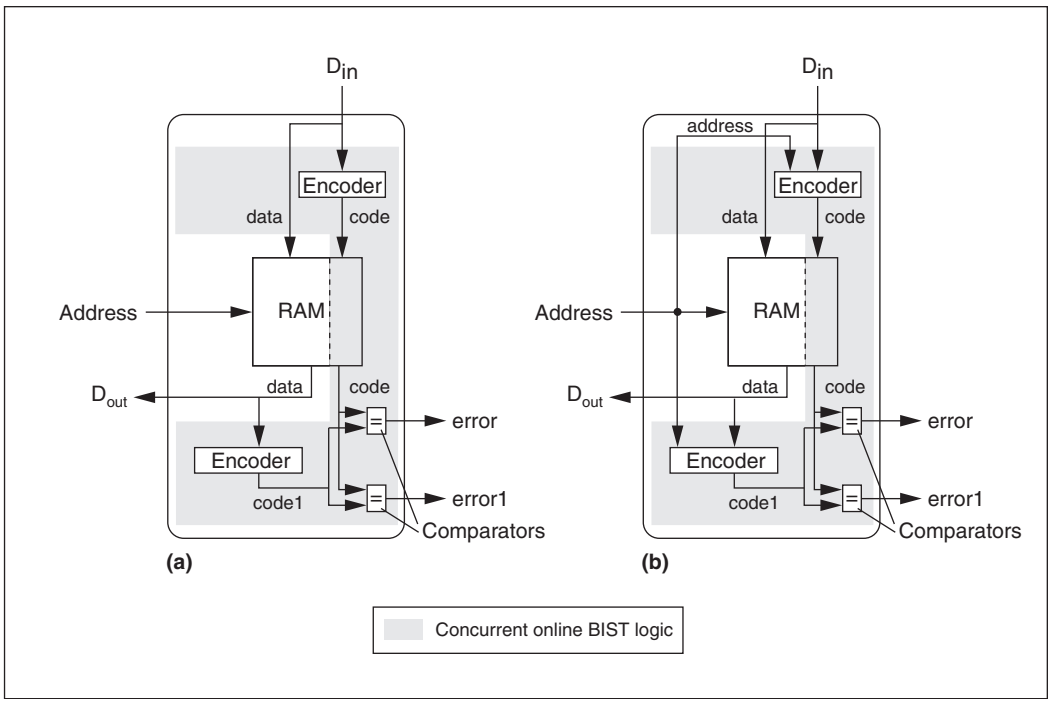

Figure 1. Online BIST architecture based on separable codes: code word including data only (a) and code word including data and address (b).

Test strategies

To meet reliability, overhead, and memory performance constraints, the architecture integrates offline, concurrent online, and nonconcurrent online BIST schemes, mixing various customized local solutions, each tackling a different problem. We adopted four main test strategies:

- To meet fault detection requirements and monitor permanent and transient faults online, we adopted code-based fault detection that applies information redundancy to data and addresses.
- The memory layout is fully customizable, so we implemented an architecture-based fault avoidance technique. As a result, the hardware implementation of the target memory allows burst errors with sizes up to the maximum detectable by the code.
- At power-up, the memory is tested offline with march tests.
- A fault latency reduction architecture lets the BIST hardware exploit idle time to further test the memory and meet strict fault tolerance requirements.

Code-based fault detection

The main fault detection feature of our architecture is implemented as a code-based solution that detects faults in the memory cells and the addressing logic.

Detecting faults in memory cells. The reliability constraints require the memory to output correct data, or at least to assert a warning signal if the data is corrupted. Our strategy is to check the correctness of the stored data when a read operation is requested. This approach relies on a backup copy of the memory content, updated with the memory and used as a reference when the memory is read. Because duplication of the entire memory would introduce an unacceptable overhead, we devised a code-based solution.

The data written into the memory is not the original data, but a code word consisting of the original data and a code computed on the data itself. We used separable codes, which distinguish the data and the code within the code word. We used the standard online memory-BIST architecture sketched in Figure 1a.

In our approach, a write operation consists of

an encode-and-write operation, and a read operation consists of a read-and-check. When writing to memory, the encoder unit encodes the incoming data and generates a code word (**data, code**) that is stored in memory. When reading memory, the system accesses the entire code word. The data part of the code word is encoded again, and the new code (**code1**) is compared with the one stored in the code word. The data is considered valid if and only if **code** and **code1** are identical; otherwise, the data is treated as faulty and an **error** signal is asserted.

The validation fails if either the comparator or the encoder is faulty, or if the code masks the faults in the addressed word. For this reason, both units should always be duplicated. As shown in Figure 1a, **code** and **code1** are computed by two different encoders and compared by two comparators.

Detecting faults in addressing logic. An address fault typically causes an unexpected cell to be accessed by a given address during both a read and a write operation.[6] In other words, because of an address fault, two different addresses point to one memory cell, or, similarly, a single address accesses two cells at the same time.

For example, consider cell C0 accessed by address A0, and C1 by A1. In the presence of an address fault, assume that A0 addresses both C0 and C1, and thus C1 can be accessed by both A0 and A1. As a fault effect, the memory generates an incorrect data output in the following two cases.

Whenever C0 is read, C1 is also read, and the resulting data is a logic combination of the content of C0 and C1. Thus, the output of the sequence {**write (C0, A0); write (C1, A1); read (C0, A0)**} is usually unpredictable. In this case, the occurrence of an address fault actually generates faulty data and thus is detected by the architecture in Figure 1a. The logical combination of the code words in C1 and C0 generates a code word in which **data** and **code** likely do not match, and the architecture in Figure 1a recognizes the data as faulty.

In the second case, when C0 is updated, C1 is overwritten with the same value. Therefore, when the sequence {**write(C1, A1); read(C1, A1)**} is interleaved by {**write(C0, A0)**}, the data read from C1 corresponds to the actual content of C0, and the data previously written on C1 by A1 is lost. To handle this case, the memory stores the address used to write the cell, to verify its correspondence with the address used to read the cell. To minimize overhead, we extended the architecture in Figure 1a to compute the code included in the code word, starting from both the incoming data and the address used for accessing the memory in writing mode (Figure 1b).

Except for code computation, the architectures in Figures 1a and 1b behave the same. The code word is updated whenever new data is written to the cell; when the memory is read, the code is recomputed (**code1**) starting from the data in the code word and the reading address. The validation fails if **code1** and **code** in the code word are different because of an address or data fault.

Code selection. The overhead incurred by this approach mainly consists of the encoder's complexity and the spare memory bits used to store the codes. We selected a suitable encoding algorithm to trade off the reliability requirements with the introduced overhead. We used Hamming error-correcting code,[7] and to minimize its size, we encoded the data and address values together. The algorithm considers a single 96-bit-wide data-bit string—80 bits for data and 16 for the address—which can be encoded with a 7-bit code. The selected code detects burst errors of up to 7 bits in the memory cells.

Architecture-based fault avoidance

We structured the fully customizable memory layout to guarantee that the targeted permanent and transient faults, occurring in either cells or addressing logic, generate burst errors of at most 7 bits in the cells.

Designers usually implement a memory array by replicating a basic memory block in columns, rows, or both. We organized our memory array by columns split into blocks equal in width to the size of the maximum detectable burst error. Thus, each block is 10K $\times$ 7 bits wide, and 13 blocks implement the 10K $\times$ 87-bit memory.

Table 1. Adopted background pattern sequences for two different patterns.

BGP0	BGP1
0000000	1111111
0000111	1111000
0011001	1100110
0101010	1010101

A faulty block causes a burst error of at most the size of the block (7 bits) in the memory cells and is therefore detectable by the code. An address fault causes access to an incorrect block, and thus the memory cells could have a sequence of 7 bits different from the expected ones. This case is also treated as a 7-bit burst error and is detectable by the code. Larger burst errors can appear only in the presence of multiple-block failures. Our experience on previous designs showed that this type of fault is unlikely to occur.

Offline testing

At power-up, the encoding mechanism is disabled and the entire memory is tested offline to detect the occurrence of permanent faults. We implemented March Test C–, which has a complexity of 10 n.[6] It detects address, stuck-at, transition, and coupling (both inversion and idempotent) faults between cell pairs. We also used background patterns (BGPs), in a march test consisting of {⇑(Wbgp0, Rbgp0, Wbgp1, Rbgp1)}, to detect intraword coupling faults—coupling faults among bits of the same cell.[8]

An 87-bits-per-word memory usually requires seven BGPs, but we shortened the test execution time significantly by taking into account the actual memory layout as defined earlier. In the column structure, intraword coupling faults occur only between bits of the same block. Therefore, each block can be considered a separate entity and its cells tested exhaustively with the BGPs listed in Table 1. Moreover, in a memory cell access, 13 cells of different blocks are accessed in parallel; applying the same BGP to all of them lets them be tested concurrently.

Figure 2 shows the BIST logic added to the architecture in Figure 1b to perform offline testing. The BIST controller manages test execution, the background pattern generator provides the BGPs, and the address generator provides the memory addresses used during test execution. Memory inputs and outputs have been multiplexed to disable encoding during offline test execution. The BIST controller controls the AG, the BGPG, and the multiplexers.

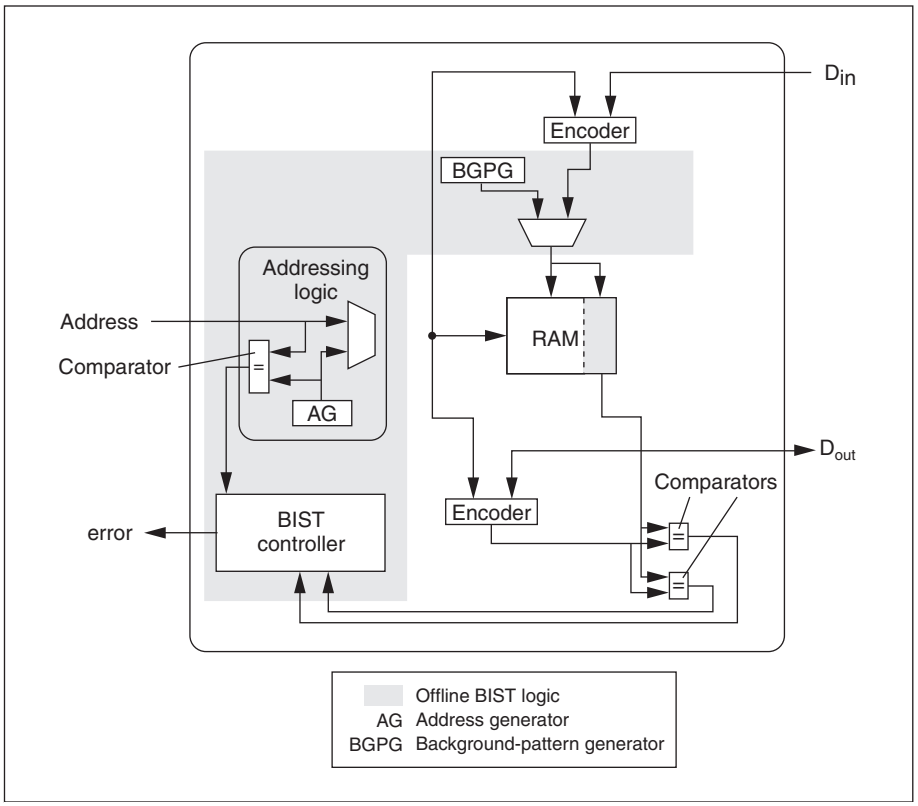

Figure 2. Concurrent online and offline BIST architecture.

Fault latency reduction architecture

We used two different approaches to meet the fault tolerance constraint and detect a fault, either in the cells or the addressing logic, within 500 memory accesses from its appearance.

First, we artificially increased the memory access rate to speed up the detection of already activated faults. The BIST controller uses memory-idle time to access the memory cells and verify the cor-

rectness of the stored code words.

In the second approach, the cells accessed during idle time are not only read and verified, the controller also tests them using a set of suitable patterns. This operation must take place without losing the content of the memory cell under test and without degrading system performance—that is, neither interrupting nor delaying normal system behavior. Therefore, we extended the architecture in Figure 2 by inserting a shadow register (S-reg) to functionally replace the cell under test. We also gave the BIST controller the capability of managing the nonconcurrent online test. Figure 3 diagrams the final offline and online BIST architecture.

The pseudocode in Figure 4 describes the nonconcurrent online test process. The cells are considered one at the time. First, the BIST hardware reads the cell and verifies its content using the code-based approach described earlier. If the cell is not faulty, its content is copied in S-reg; then the encoding is disabled and the entire cell is tested with the BGPs described in Table 1. To shorten the fault latency, the cell is tested each time using only one BGP; when it has been tested four times, all the BGPs have been applied.

Whenever a read or write operation is requested, the BIST controller suspends test execution and serves the request. If the operation is requested on the cell under test, the BIST controller transparently executes the operation using S-reg instead of the actual memory cell. At test completion, the content of S-reg is copied back to the cell, and another cell is set under test.

In the case of a faulty memory, the BIST controller asserts the error signal, and the system waits for user intervention. The simple BISR strategy replaces the faulty cell using S-reg in the normal memory mode. The BISR results in graceful degradation; it repairs the faulty memory cell, but it stops the nonconcurrent online BIST.

The BIST logic added to meet fault latency requirements is not critical. The occurrence of a fault in it would increase fault latency without decreasing fault coverage. If the faulty BIST

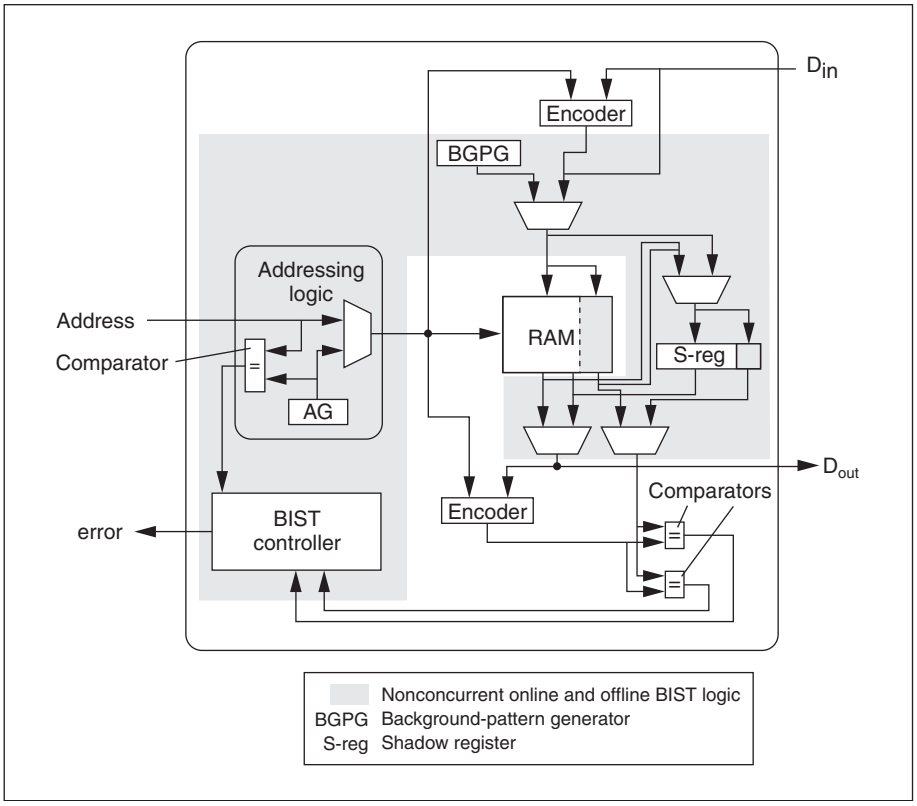

Figure 3. Final offline and online BIST architecture.

```
foreach BackgroundPattern bgp {
   foreach memory address i {
      --Read & verify the data
      --The symbol & represents the VHDL concatenation
       operator
      read DATA [i] &CODE [i]
      if encoding (DATA [i]) ≠ CODE [i]
         { ERROR }
      --Save the data
      S-REG = DATA [i] &CODE [i]
      --Write bgp0
      DATA [i] &CODE [i] = BGPG (bgp, 0)
      --Read bgp0
      if (DATA [i] &CODE [i] ≠ BGPG (bgp, 0) )
         { ERROR }
      --Write bgp1
      DATA [i] &CODE [i] = BGPG (bgp, 1)
      --Read bgp1
      if (DATA [i] &CODE [i] ≠ BGPG (bgp, 1) )
         { ERROR }
      --Restore the original data
      DATA [i] &CODE [i] = S-REG
   }
}
```

Figure 4. Nonconcurrent online test algorithm.

Table 2. IP core's area overhead.

Unit	Equivalent gates	Relative area (%)
Original memory	370,976	90.59
Extra memory bits for code storage	32,460	7.94
Concurrent online BIST logic	2,146	0.52
Nonconcurrent online BIST logic	1,814	0.44
Offline BIST logic	1,195	0.29
TAP controller	914	0.22

logic fails to detect faults via nonconcurrent online testing, the same faults will be detected by concurrent online testing. If the latency degradation is unacceptable for an application, the BIST logic must be made fail-safe.[9] Our target application didn't require fail-safeness.

Final implementation

We enhanced the final circuit to meet some additional requirements. To guarantee accessibility of all units during the end-of-production test, we synthesized the circuit using full-scan methodology. In addition, we designed a custom test access port controller[10] to fully control the BIST logic of the implemented testing strategies. The TAP controller

- activates BIST;
- interrupts BIST when it detects an error (BIST executes the test algorithm, and when a fault is detected, an external tester can locate the fault's exact position through the scan chains);
- restarts BIST after a fault detection; and
- disables BIST.

We synthesized the BIST architecture with LSI Logic's G11 technology. Table 2 describes the IP core's area in terms of equivalent gates and percentages of the original memory for the implemented test strategies and the TAP controller.

Although developed for a specific application, this BIST architecture can be customized for the requirements of other applications, which often have less-constraining reliability requirements. ■

References

1. M. Nicolaidis and Y. Zorian, "Online Testing for VLSI—A Compendium of Approaches," *J. Electronic Testing, Theory and Applications*, vol. 2, nos. 1/2, Feb.-Apr. 1998, pp. 7-20.
2. A.J. van de Goor, *Testing Semiconductor Memories: Theory and Practice*, John Wiley & Sons, Chichester, UK, 1991.
3. H. Al-Asaad, B.T. Murray, and J.P. Hayes, "Online BIST for Embedded Systems," *IEEE Design & Test of Computers*, vol. 15, no. 4, Oct.-Dec. 1998, pp. 17-24.
4. A. Benso et al., "An On-line BISTed RAM Architecture with Self-Repair Capabilities," to be published in *IEEE Trans. Reliability*, Sept.-Oct. 2001.
5. A. Benso et al., "An On-line BISTed SRAM IP-Core," *Proc. IEEE Int'l Test Conf.*, IEEE CS Press, Los Alamitos, Calif., 1999, pp. 993-1000.
6. A.J. van de Goor, "Using March Tests to Test SRAMs," *IEEE Design & Test of Computers*, vol. 10, no. 1, Mar. 1993, pp. 8-14.
7. F.J. Macwilliams and N.J.A. Sloane, *The Theory of Error-Correcting Codes II*, *North-Holland Mathematical Library*, vol. 16, North-Holland, Amsterdam, 1998.
8. A.J. van de Goor and I.B.S. Tilli, "March Tests for Word-Oriented Memories," *Proc. IEEE Design Automation and Test in Europe*, IEEE CS Press, Los Alamitos, Calif., 1998, pp. 501-508.
9. P.K. Lala, *Self-Checking and Fault-Tolerant Digital Design*, Morgan Kaufmann, San Mateo, Calif., 2001.
10. *IEEE Std 1149.1-1990, Test Access Port and Boundary-Scan Architecture*, IEEE Press, Piscataway, N.J., 1993.

Alfredo Benso is a research assistant in the Computer and Information Sciences Department of the Politecnico di Torino, Italy. His research interests include design-for-testability techniques, BIST for complex digital systems, dependability analysis of computer-based systems, and software-implemented hardware fault tolerance. Benso has an MS in computer engineering and a PhD in computer engineering, both from Politecnico di Torino. He chairs the Web-Based Activities

Group of the IEEE Computer Society's Test Technology Technical Council (TTTC).

Silvia Chiusano is a research assistant in the Computer and Information Sciences Department of Politecnico di Torino. Her research interests include high-level testing, design-for-testability techniques, BIST, and dependability. Chiusano has an MS and a PhD, both in computer engineering, from Politecnico di Torino.

Giorgio Di Natale is pursuing a PhD in the Computer and Information Sciences Department of Politecnico di Torino. His research interests include design-for-testability techniques, built-in self-repair, and FPGA testing. Di Natale has an MS in computer engineering from Politecnico di Torino. He is an associate Webmaster of the IEEE Computer Society's Test Technology Technical Council.

Paolo Prinetto is a full professor of computer engineering at Politecnico di Torino and an adjunct professor at the University of Illinois at Chicago. His research interests include testing, test generation, BIST, and dependability. Prinetto has an MS in electronic engineering from Politecnico di Torino. He is a Golden Core member of the IEEE Computer Society and the chair of the IEEE Computer Society's Test Technology Technical Council.

Monica Lobetti Bodoni is a design-for-testability engineer at Siemens Information and Communication Networks (formerly Italtel) in Milan, Italy. Her research interests include memories, logic BIST, automatic test-pattern generation methods for systems on chips, boundary scan, and custom at-speed solutions for interconnection testing. Bodoni has an MS in nuclear engineering from Milan Polytechnic. She is an IEEE member and chair of the TTTC Board Test Technical Activity Committee.

■ Direct questions or comments about this article to Alfredo Benso, Politecnico di Torino, Dipartimento di Automatica e Informatica, Corso Duca degli Abruzzi 24, I-10129 Torino TO, Italy; benso@polito.it.

For further information on this or any other computing topic, please visit our Digital Library at http://computer.org/publications/dlib.