Programmable built-in self-testing of embedded RAM clusters in system-on-chip architectures

Programmable built-in self-testing of embedded RAM clusters in system-on-chip architectures

Authors: Benso A., Di Carlo S., Di Natale G., Lobetti Bodoni M., Prinetto P.,

Published in the IEEE COMMUNICATIONS MAGAZINE Vol. 41 No. 9, 2003.

DOI: 10.1109/MCOM.2003.1232242

Programmable Built-In Self-Testing of Embedded RAM Clusters in System-on-Chip Architectures

A. Benso, S. Di Carlo[1], G. Di Natale, P. Prinetto
Politecnico di Torino (Dipartimento di Automatica e Informatica)
Torino, Italy

M. Lobetti Bodoni
Siemens Mobile Communications
Cinisello Balsamo, Milano, Italy

Abstract— **Multi-port memories are widely used as embedded cores in all communication System-on-Chip devices. Due to their high complexity and very low accessibility, Built-In Self-Test (BIST) is the most common solution implemented to test the different memories embedded in the system. This paper presents a programmable BIST architecture, based on a single micro-programmable BIST Processor and a set of memory Wrappers, designed to simplify the test of a system containing a large number of distributed multi-port memories of different sizes (number of bits, number of words), access protocol (asynchronous, synchronous), and timing.**

I. INTRODUCTION

Silicon area is now so cheap and integration technologies so advanced that industries can embed in a single chip, usually referred to as System-on-Chip (SoC), all the components and functions that historically were placed on a hardware board. Each component or function is now available as a pre-designed complex functional block, or *embedded core*.

Embedded memories are the most dense components within a SoC, accounting for up to 90% of its real estate. Today's technologies allow the design and manufacturing of memory cores with many I/O ports, and multi-port RAM core generators are commonly available in many ASIC vendors library as LSI-Logic, Texas Instruments and ST Microelectronics. To have an idea of today's SoC complexity, it is enough to consider that typically more than 30 embedded memories are placed on a single chip, they are scattered around the device rather than concentrated in one location, they all have different types, sizes, and access protocols and timing, and they can even be doubly embedded inside embedded cores. From a testability point of view, memories also are the most sensitive to process defects, making it essential to thoroughly test them in the SoCs.

[1] Contact Author: Stefano Di Carlo, Politecnico di Torino (Dipartimento di Automatica e Informatica), Corso Duca Degli Abruzzi 24, I-10129, Torino, Italy phone: +39-011-564-7080; fax: +39-011-564-7099; e-mail: stefano.dicarlo@polito.it

This new design philosophy based on the use of embedded cores, leads to a radical change in the test engineering process. First of all, direct accessibility to interconnections and cores' boundaries is not possible, nevertheless test patterns and test responses have still to be delivered to the core or to the SoC boundaries.

In the case of memory cores, the test methodology of choice is Built-In Self-Test (BIST). BIST offers a simple and low-cost means to test for failures of embedded memories without significantly impacting the device performance. In this scenario, the implementation of an efficient BIST strategy for SoCs including several multi-port RAMs requires to take into account the different sizes (number of bits, number of words), access protocol (asynchronous, synchronous), and timing of the memories embedded in the system, to minimize the BIST area and routing overhead and to fulfill power budget constraints. Moreover while it has been used primarily for production pass/fail testing, BIST should be extended to provide the diagnostic data required for process monitoring and repair. A successful BIST for embedded memories has to guarantee *core accessibility*, *scalability*, *In-System Programmability* (*ISP*), *low overhead*, and *flexibility in the test scheduling*.

This paper presents the efforts and the results obtained in designing a proprietary BIST architecture to tackle the above-mentioned set of problems.

The paper is organized as follows: Section II summarizes some of the most significant memory BIST architectures presented in literature; Section III gives a general overview of the proposed BIST architectures whereas Section IV and Section V detail the structure of the main blocks of the architecture. In Section VI and Section VII the scheduling and diagnosis facilities of the proposed BIST are detailed, and in Section VIII a possible optimization is discussed. Section IX presents a real application of the proposed approach on an industrial case study, and finally Section X summarizes the main contributions of the work and concludes the paper.

II. State of The Art

Several memory BIST solutions have been proposed to test both single and multi-port memories [1] [2], static and dynamic memories [3] [4]. Programmable memory BIST has been proposed in [5] and [7] [14] to increase flexibility in applying different combinations of test patterns targeting different types of faults. Despite their effectiveness, all these solutions are designed to address the problem of testing a single type of memory, and none focuses on the problem of concurrently testing several heterogeneous embedded memory arrays. This problem has been addressed in [6] and [8], where the authors propose a built-in self-diagnostic method to simultaneously diagnose spatially distributed memory modules with different sizes. The approach is based on the *serial interfacing technique* proposed in [9]. The basic idea is to synthesize the I/O port of each buffer as a scan chain from which the test patterns can be provided and memory contents can be read. The solution is very easy to implement but it is not so efficient in

terms of test speed and area overhead, and it does not take into account power consumption constraints. Moreover, all the memory tested in parallel must be of the same type. A deterministic BIST state machine was designed in [10] to test multiple RAMs with different characteristics. Although all the memory modules are tested (truly) concurrently, each memory module receives its own control signals from the BIST controller. This solution has the disadvantages of large routing area overhead and a complex design of the BIST controller.

III. Memory BIST Management Architecture

The goal of this paper is the design of a proprietary BIST scheme to tackle the problem of testing the memory subsystem of a complex SoC. Figure 1 gives an overview of the proposed BIST architecture.

A single *BIST Processor* is in charge of performing the test of all (or a subset of) the memories of the system. Using a minimal set of *Communication Signals*, the BIST Processor coordinates, executes, and synchronizes the test algorithm of the memories under test. The BIST Processor is μ-programmable: the test algorithm is stored as a sequence of elementary test primitives in a dedicated memory (μ-program memory); these instructions include (but are not limited to) the update of the address generators, the application of a test pattern, the comparison of a memory cell with an expected value. This solution allows, if necessary, to program the system at run-time to execute any required test algorithm. The BIST Processor functionalities and communication protocol are independent from the number and characteristics of the memories embedded in the system.

The different test primitives that constitute the test algorithm are received by a Wrapper placed around each memory. In particular, each wrapper is composed of a set of *Port-Wrappers* (one per each memory port) and a *Dispatcher*. The *Port-Wrapper* contains the standard blocks required to implement the BIST capabilities (i.e., an address generator, a pattern generator, and a comparator), and an interface block designed to translate the communications received from the BIST Processor into the particular memory access protocol. The *Dispatcher* is in charge of collecting the test commands from the BIST Processor, and delivering them to the different Port-Wrappers.

Finally, two *scan chains* are used to connect all the Port-Wrappers to allow respectively the scheduling of the memory under test and full diagnosis capability.

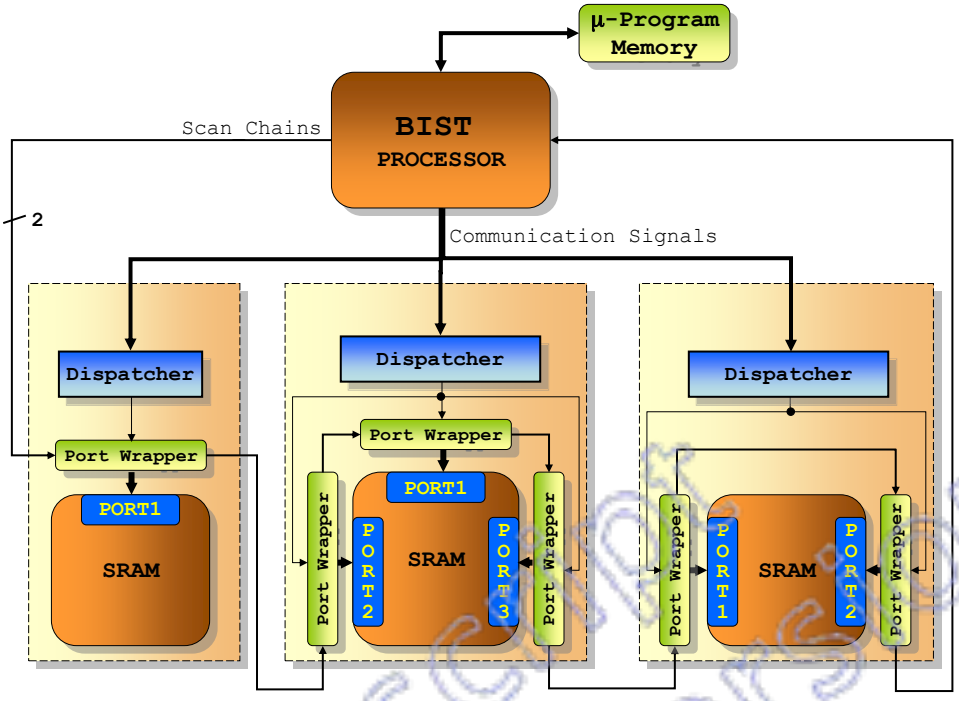

Figure 1: Basic Architecture

The proposed scheme guarantees the following goals:

- *Core accessibility*: the test of each memory is controllable and observable using a minimal set of communication signals, including only two *scan chains* connecting all the Port-Wrappers;
- *Scalability*: the BIST Processor implementation is independent from the number and characteristics of the memories embedded in the system;
- *In-System Programmability* (*ISP*): implementing the BIST Processor as a µ-programmable machine provides the test engineer a flexible and reusable block, which can be used to manage the BIST of any number of memories of any size, and it is independent from the test algorithm, which can be dynamically downloaded in the BIST Processor itself;
- *Low overhead*: using a single BIST processor and a minimum set of communication signals allows minimizing, with respect to a traditional BIST solution where each memory has a dedicated BIST controller, the area overhead and the connectivity around each RAM;
- *Flexible test scheduling*: the set of memories to be tested can be freely selected by the test engineer, using either test primitives stored in the test program, or a dedicated scan chain to properly set a status bit in each memory. Moreover, the proposed solution allows running concurrently the BIST of a set of memories with different number of ports, sizes, access protocols, and timing.

The following sections will further detail the blocks composing the architecture.

IV. THE BIST PROCESSOR

The proposed memory BIST is based on a single BIST Processor used to test all the memories of the target SoC. To increase flexibility, the BIST execution is based on a μ-programmable approach. Due to their regular structure, the most popular and widely accepted deterministic test algorithms for memory BIST are March Tests. A March test is a finite sequence of operations (March elements) applied to each memory cell in the memory array in either ascending or descending order before proceeding to the next cell [1]. March tests are popular because of their low temporal complexity, regular structures, and their ability to detect different types of faults. The proposed BIST Processor has therefore been optimized to implement March Tests. The chosen algorithm is stored in a dedicated *μ-Program memory*, coded through a set of *test primitives*. The μProgram-Memory can be either a ROM or an In-System Programmable device. In the former case, the test program is fixed at design time, whereas in the latter one any custom test algorithm can be downloaded into the μ-Program memory at test time.

After selecting the set of memories under test, the BIST Processor reads from the μ-Program memory one test primitive at a time, forwards it to all the Wrappers of the memories under test, and waits until its completion by all the target memories.

When the test program is completed (i.e., all the test primitives have been applied), the BIST Processor reads the test results from each memory. If a fault is detected, the faulty memories can be located resorting to a set of diagnosis capabilities (See Section VII).

The architecture of the BIST Processor and the μ-Program memory are strongly influenced by the peculiar characteristics of multi-port memories. In fact, due to the possibility of concurrently accessing several cells, new fault models must be targeted [11] and ad-hoc March Algorithms must be adopted to cover these new types of fault. In particular, the proposed implementation is optimized to implement March Algorithms for multi-port memories presented in [12]. The main characteristic of these algorithms is the use of nested cycles to access the different memory ports:

$$\left\{\Uparrow_A \left(\Uparrow_{B=0}^{A-1} \left(\Uparrow_{C=B+1}^{n} \ldots\ldots\right)\right)\right\}$$

where $\Uparrow_{B=0}^{A-1} \Uparrow_{C=B+1}^{n}$ denotes a nested addressing sequence in which cell B goes from 0 to $A - 1$ and, for each value of B, cell C goes from $B + 1$ to n. A pseudo-C code of this nested addressing sequence would correspond to two nested *for* cycles:

```
for (B=0;B<A-1;B++)
for (C=B+1; B<n; B++)
      ……
```

As previously explained, each step of the test-program is coded in the μ-Program memory as a sequence of test primitives, one for each memory port. The set of *test primitives* needed to implement the proposed family of March Algorithms are:

- W0: Write pattern
- W1: Write not(pattern)
- R0: Read and verify a pattern
- R1: Read and verify a not(pattern)
- INC: Increment the address generator and define the end of a March Element
- DEC: Decrement the address generator and define the end of a March Element
- INCCOND: Conditionally increment the address generator
- DECCOND: Conditionally decrement the address generator
- SUB: Increment the address generator of x, with x>1
- ADD: Decrement the address generator of x, with x>1
- LOAD: Load a value in the address generator
- NME: New March Element
- NOP: No Operation
- NEXTP: Next Pattern
- CONF: Define the set of SRAM under test
- RUN: Synchronization primitive
- END: End of test

The external interface of the BIST Processor can be designed in order to match the target system requirements. Possible solutions are a P1500 compliant interface, an addressable device on the system bus, or a JTAG interface, as in the case study presented in Section IX.

V. The Memory Wrapper

The Wrapper placed around each memory has to execute the test primitives broadcasted by the BIST Processor regardless the particular memory access protocol. The Wrapper is therefore the only element in the architecture taking care of the number of ports, the size, and the access protocol of the memory it wraps.

The Wrapper generates the correct test patterns and memory addresses required to execute the received test primitives, and compares the values read during the test with the expected ones.

The Wrapper architecture consists of a *Dispatcher* and a set of *Port Wrappers.*

A. Dispatcher

Each RAM under test has a dedicated dispatcher, which receives the test primitives for all the Port Wrappers from the BIST Processor. Since the primitives are sent sequentially but they have to be applied at the same time in order to execute the required operations concurrently on all the ports of the memory, each dispatcher saves all the primitives in a temporary register and delivers them to each port wrapper only after receiving a synchronization test primitive (RUN). This solution allows to dramatically reducing the routing overhead that would be required to send all the primitives in parallel using for each port a dedicated bus.

B. Port Wrapper

Each memory port has a dedicated *Port-Wrapper* that generates the test patterns (address and data) and verifies the correct behavior of the memory according to the primitive received from the dispatcher. The result of each primitive is signaled on an output line.

The internal structure of a Port Wrapper is drawn in Figure 2. The *Address Generator (AG)* is in charge of generating the correct address where the test pattern, provided by the *Pattern Generator (PG)*, has to be written or verified. The PGs can be easily customized in order to target different fault types [13]. Its implementation is nevertheless always very simple, and never more complex than an up-counter. The correctness of the content of a memory cell is evaluated using a simple C*omparator.*

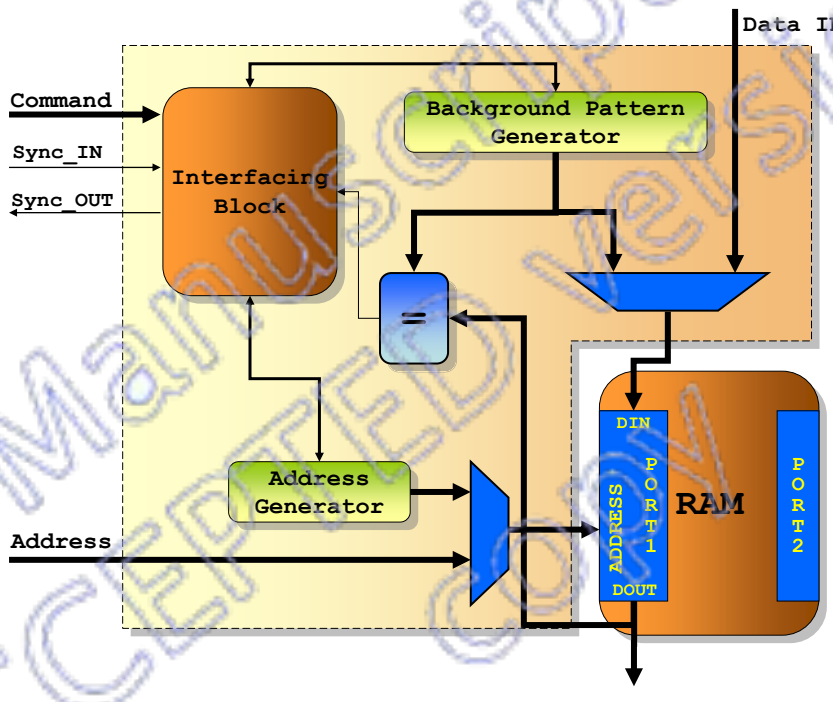

Figure 2: Port Wrapper architecture

Two Status Bits are used respectively to set the memory in *transparent* or in *test mode* (the *Mode Status Bit*) and to store the *test results* at the end of the BIST algorithm (the *Result Status Bit*). All the memories set in test mode are tested in parallel, whereas those set in *transparent mode* are bypassed and not tested; this feature is required to allow a flexible scheduling of the memories under test. To set and read them, the status bits of all the Port Wrappers are dynamically connected into a global scan chain.

Finally, each Port Wrapper includes an *Interfacing Block* able to receive the test primitives (*Command*) from the Dispatcher and to execute them on the memory using the required protocol. Moreover, the Interfacing Block receives a synchronization signal (*Sync_IN*) from the previous port wrapper, and produces an *output synchronization signal* (*Sync_OUT*) needed by the other wrappers and by the BIST Processor to synchronize the scheduling of the next test primitive.

The Sync_IN signal of each Port Wrapper is directly connected to the Sync_Out signal of the previous one, except for the last Port Wrapper whose Sync_OUT signal is connected to the BIST Processor. The *Sync_OUT* signal is enabled only when the Sync_OUT signal of the previous Port Wrapper is asserted. Therefore, the BIST Processor receives the logic-AND of the output signals generated by all the Port Wrappers.

From a functional point of view, *Sync_OUT* assumes different meanings depending on the received test primitive. As an example, in case of a Read or Write operation, it has the meaning of *End of Instruction (EOIN)*. It is asserted when the memory actually ends the execution of the command. This mechanism guarantees the synchronization among memories with different timing and access protocols. In case of a primitive to increment or to decrement the value of the address generator, Sync_OUT has the meaning of *End of Address (EOAD)*. It is asserted when the addressing space has been visited by the address generator, allowing the synchronization among memories with different sizes.

Two types of Port Wrappers are available: one for the first port of each memory and one for the others ports. The main difference between the two lays in the fact that the Port Wrapper connected to the first port of the memory implements the main addressing loop of the march tests family discussed in Section IV, whereas the addresses applied to the memory by Port Wrappers connected to the remaining ports are relative to the value of the address generated by the previous Port Wrapper.

In order to minimize the routing overhead, the signals exchanged between the BIST Processor and the memory Wrappers (command signals, synchronization signal, scan chain signals) are multiplexed. In particular, these signals are multiplexed at the port-wrapper level. All the information is routed using only 6 signals (4 command signals and 2 synchronization signals).

VI. TEST SCHEDULING

An important issue to be faced when running concurrently the BIST of several modules is fulfilling power budget constraints. In fact, BIST typically results in a circuit activation rate higher than the normal one, and an over-dissipation of power may seriously damage the devices. Moreover, the variety of memories that can be found in a complex architecture may require different test algorithms. To address these two issues, the proposed approach implements a very flexible scheduling mechanism. In particular, it is possible to select the set of memories to be tested using either a dedicated test primitive as part of the test algorithm, or setting the *Mode Status Bit* flag into the memory Wrapper through a scan chain (See Section V). Only the Wrappers of the selected memories will execute the test primitives received from the BIST Processor; all the other ones will be set in transparent mode and therefore bypassed. In this way, several test algorithms may be stored in the μProgram-Memory and may be applied sequentially to different sets of memories. The definition of algorithms or guidelines for the

selection of the best scheduling is a task that depends on the particular target system and it is therefore out of the scope of this paper. Our main focus has been in the design of an architecture that allowed a flexible definition of the test scheduling. The two mechanisms implemented to allow the scheduling of the memories under test are briefly explained in the following.

A. Scheduling using the "CONF" primitive.

Using the CONF primitive, it is possible to embed scheduling information into the test Program. The representation of this primitive in the µProgram-Memory is defined as follows:

- The CONF opcode;
- The number of 4-bit words used to code the *ActivationMask*
- The ActivationMask, i.e., a mask of bits, where each bit corresponds to one memory in the system. To include a memory in the set of the SRAMs under test, the corresponding bit in the ActivationMask has to be set.

As an example, let's consider the system in Figure 3:

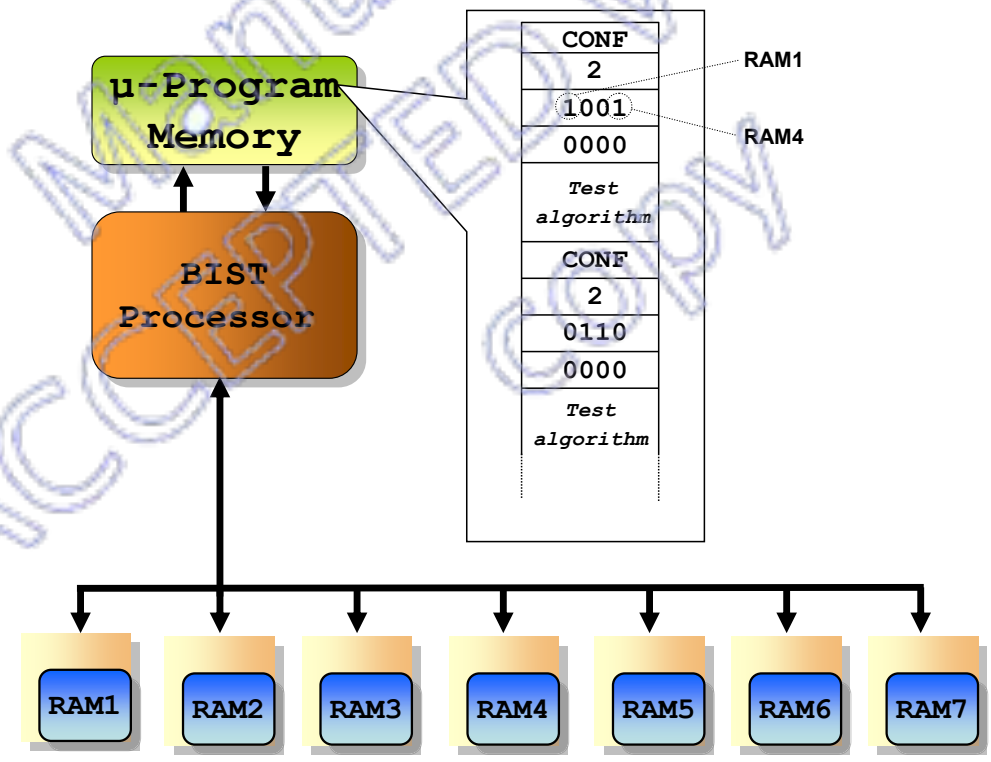

Figure 3: Scheduling using the "Conf" primitive.

When the BIST Processor reaches a CONF primitive during the Test Program execution, it reads the ActivationMask and configures all the memory wrappers using the scan chain defined in Section III in order to activate the required scheduling plan. The first ActivationMask described in Figure 3 sets RAM1 and RAM4 under test, whereas the second one sets RAM2 and RAM3 under test. In order to

define different test sessions and to collect test results, at the end of each algorithms the BIST processor stops the test program execution and waits for a new start primitive to continue with the next one.

B. Scheduling using the Scan chain option.

In order to give the designer a higher flexibility, the set of memories under test can be also set loading the appropriate ActivationMask directly from the outside using a scan chain protocol. In order to jump to the appropriate test algorithm in the μ-program memory, also the starting value of the *μ-program memory Address Register* can be loaded in the BIST processor using the same protocol.

VII. DIAGNOSIS

Fail map extraction is required to output the relevant data necessary to determine why a failure occurred within a memory. This data is post-processed using diagnostic software to isolate the defective memory and location within the memory. Therefore, when a faulty memory is detected, the proposed approach allows collecting diagnostic information about the location of the faulty memories, the ports where the fault has been detected, the addresses of the faulty cells, and the detecting patterns. This information is stored into the *Result Status Bit*, the *Address Generator*, and the *Background Pattern Generator* of each Port-Wrapper and can be scanned out via the *Results_Scan_Chain*. To allow even more detailed diagnostic capabilities, it is also possible to include in the Result_Scan_Chain the test primitive that triggered the detection of the fault. To reduce the scan chain length, depending on the result of the test (*Result_Status_Bit*), each Port-Wrapper configures its portion of the *Results_Scan_Chain* in one of the following two ways (Figure 4):

- *Result_Status_Bit='1'*: the memory is not faulty; only the *Result_Status_Bit* is placed on the scan chain;
- *Result_Status_Bit='0'*: the memory is faulty; the *Result_Status_Bit* is chained to the content of the Address Generator and the Background Pattern Generator.

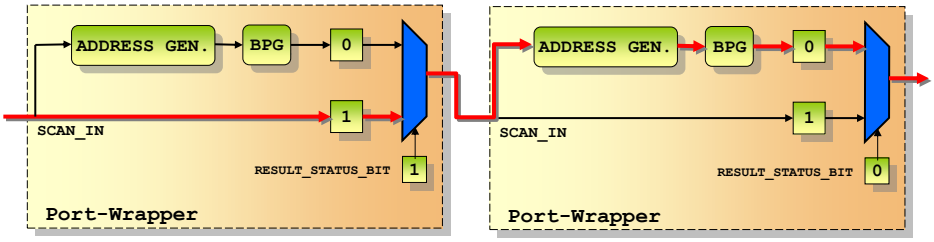

Figure 4: Results_Scan_Chain

VIII. FURTHER OPTIMIZATION

To further reduce the BIST area overhead, the designer can share a single Wrapper for a cluster of identical memories (same type, width, and size) to be tested in parallel.

This optimization is made at the Port-Wrapper level. For each Port-Wrapper only one Address Generator and one Background Pattern generator are needed. The only difference with the previously described Port-Wrapper structure is that a shared Port-Wrapper contains a pair of Status Bits and a comparator for each memory. In this way, when a fault is detected, the Result Status Bit of the faulty memory is set, the memory is disconnected, and the Wrapper keep on testing the remaining memories of the cluster. Obviously, in this case, the status of the Address Generator and the Pattern Generator of the faulty memory are not pre rved. To collect diagnostic information, the test must be re-executed on the faulty memory only by properly setting its Mode Status Bit.

Finally, since a fault in the BIST logic can be detected only if it causes an error that is detectable as a memory fault by the test algorithm, the stuck-at fault coverage cannot be precisely computed a-priori and it will be anyway quite low. Therefore, to allow high fault coverage at the end-of-production, the BIST logic can be synthesized and tested using full-scan.

IX. CASE STUDY

A case study has been used to evaluate the proposed approach and to gather experimental results. The target circuit, named VC12AD, is a part of a telecommunication ASIC designed by Italtel SpA. Both Italtel SpA and Siemens ICN have also used the same circuit as a benchmark for the evaluation of commercial BIST Insertion Tools. The target circuit has been described in VHDL and synthesized using the *G10 LSILogic™ library*, which provides a set of RAMs of different sizes.

The VC12AD counts up to 860K equivalent gates (excluding RAMs), plus 36 small-sized RAMs, for a total of 14,704 bits and 380,503 equivalent gates.

The case study aims at evaluating the BIST architecture complexity when applied to a set of memories with very different characteristics, and the area overhead after the BIST insertion.

The 36 RAMs of the circuit are grouped into four distinct macro-areas whose characteristics are listed in Figure 5.

A. BIST architecture

In the definition of the BIST architecture, we tried to minimize the number of wrappers resorting, whenever possible, to clusters of memories (see Section VIII). As a consequence:

- Within C12A, the 2 modules tpa21x8 and the 2 modules spa*21x26 are treated as two *clusters*;
- Within C12D, the 2 modules spa21x34 and the 2 modules spa* are treated as two *clusters*;
- Within SYNDES, the memories are organized in four *clusters* of 7, 7, 6, and 1 element, respectively.

The memory clustering has been strongly influenced by the actual floor plan: for example, the 3 spa21x34 memories (2 located inside C12D and 1 in PDH_INT) are too far to be included in a single *cluster*.

The overall VC12AD structure after the BIST insertion is in Figure 5.

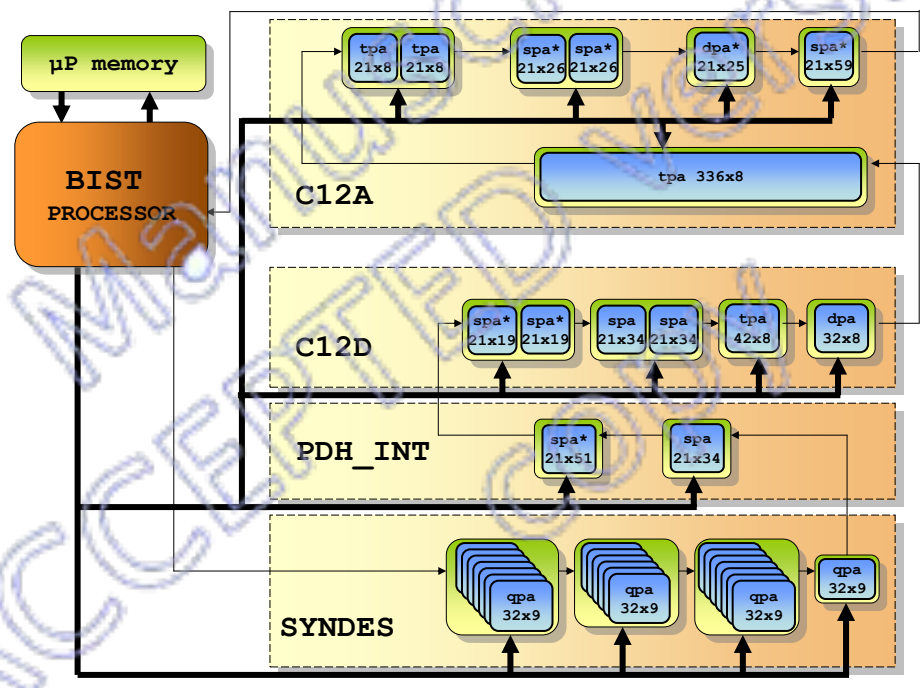

Figure 5: VC12AD BIST Architecture

B. BIST Scheduling

Due to the different characteristics of the VC12AD memories (read/write ports, read-only ports, and write-only ports are present), it is not possible to adopt a single March Algorithm for all of them. We therefore organized the BIST in four sessions, each one executing a different March algorithm:

- *Session 1:* All the single port RAMs are tested concurrently;
- *Session 2:* All the dual port RAMs are tested concurrently;
- *Session 3:* All the triple port RAMs are tested concurrently;

- *Session 4:* All the quadruple port RAMS are tested concurrently.

C. *Experimental results*

The total area overhead and the area occupation of each Wrapper and its functional blocks are summarized in Figure 6.

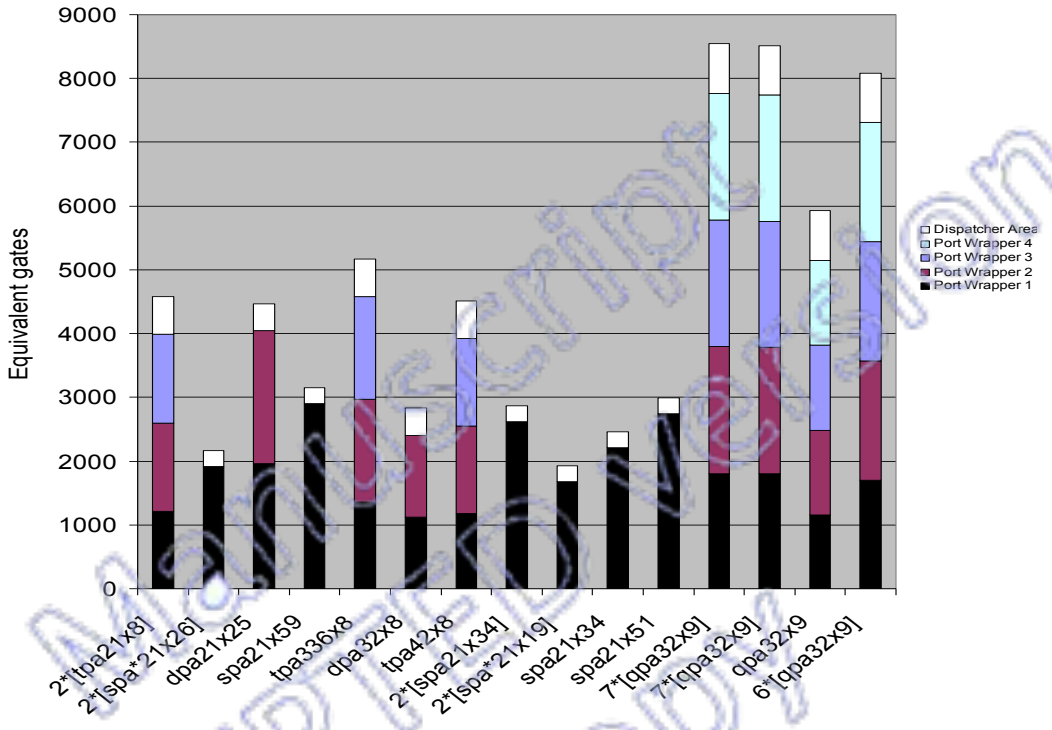

Figure 6: Wrappers area overhead

The total area overhead introduced by the port wrappers is 68,177 equivalent gates. This area is not proportional to the number of memory ports, but depends more on the port sizes and functionalities.

The BIST processor and the μ-program memory area overheads, 5,431 and 4,459 equivalent gates respectively are a fixed contribution and they are not influenced by the number of memories present in the system.

The total area overhead is, in this case study, the 17.02%. Although this result may seem quite high, it is necessary to consider that the target circuit has a lot of small memories and therefore the overhead introduced by the wrapper is significant. With larger memories the overhead would be much lower.

The area overhead introduced by a commercial BIST insertion tool has been the 22.5%.

X. CONCLUSIONS

In this paper we presented a proprietary solution for a particular industrial scenario, in which it is necessary to define the BIST strategy of a complex communication SoC including several multi-port memories of different sizes, access protocol, and timing. The proposed architecture consists in a single

BIST Processor, implemented as a μ-programmable machine and able to execute different test algorithms, a Wrapper for each memory (or cluster of memories), each Wrapper including one *Port-Wrapper* for each memory port and a special block named *Dispatcher*. Each *Port-Wrapper* contains standard memory BIST modules, and an interface block to manage the communications between the memory and the BIST Processor. The *Dispatcher* collects the instruction from the test processor and delivers them to the *Port-Wrapper*s. The proposed scheme presents several advantages. It allows running concurrently the BIST of a set of memories of different number of ports, sizes, and access protocols, minimizing the BIST area overhead and connectivity around each memory. In addition, the set of memories to be tested can be freely selected by the designer, as well as the test algorithm to be executed on each set.

The proposed memory BIST architecture deals with memory modules only. If additional modules (e.g., random logic, legacy cores, etc.) have to be BISTed as well, more complex and sophisticated approaches will have to be adopted.

ACKNOWLEDGMENTS

This work is partially supported by Istituto Superiore per le ICT Mario Boella under contract Test DOC: Quality and Reliability of Complex SoC.

REFERENCES

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

[2] Y. Wu, S. Gupta, *"Built-in self-test for multi-port RAMs"*, IEEE, Asian Test Symposium, Nov 1997, pp. 398-403

[3] M. H. Tehranipour, Z. Navabi, S. M. Fakhraie, *"An efficient BIST method for testing of embedded SRAMs*", IEEE, International Symposium on Circuits and Systems, 2001, Volume 5, pp. 73-76

[4] Chih-Tsun Huang, Jing-Reng Huang, Chi-Feng Wu, Cheng-Wen Wu, Tsin-Yuan Chang, *"A programmable BIST core for embedded DRAM"*, IEEE, Design & Test of Computers, Volume 16, Issue 1, Jan-Mar 1999, pp. 59-70

[5] H. Koike, T. Takeshima, M. Takada, *A BIST scheme using microprogram ROM for large capacity memories,* IEEE, International Test Conference, Sept. 1990, pp. 815 -822

[6] W.B. Jone, D.C. Huang, *"A parallel built-in self-diagnostic method for embedded memory arrays"*, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Volume 21, Issue 4, April 2002, pp. 449-465

[7] T. Ching-Hong, W. Cheng-Wen, *"Processor-programmable memory BIST for bus-connected embedded memories"*, IEEE, Design Automation Conference, 2001, pp. 325 -330

[8] W.B. Jone, D.C. Huang, S.C. Wu, K.J. Lee, *"An efficient BIST method for distributed small buffers"*, IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Volume 10, Issue 4, Aug 2002, pp. 512 -515

[9] B. N. Dostie, A. Silburt, and V. K. Agarawal, *"Serial interfacing technique for embedded memory testing"*, IEEE Design Test Computers, pp. 52–63, Apr. 1990.

[10] L. Ternullo, R. Dean Adams, 3. Connor, G. S. Koch., *"Deterministic Self-Test of a High-Speed Embedded Memory and Logic Processor Subsystem"*, IEEE, Proc. of International Test Conf., 1995, pp. 33-44

[11] S. Hamdioui, A. J. Van de Goor, "*Consequences of Port Restrictions on Testing Two-Port Memories"*, IEEE, International Test Conference, 1998, pp 63-72

[12] M. Nicolaidis, V. Castro Alves, H. Bederr, "*Testing Complex Couplings in Multiport Memories"*, IEEE, Transaction on VLSI systems, 3(1), March 1995, pp. 59-71

[13] A.J. van de Goor, I.B.S. Tlili, *March tests for word-oriented memories*, IEEE, Design Automation and Test in Europe Conference, 1998, pp. 501-508

[14] K. Zarrineh, S. J. Upadhyaya, *"On Programmable Memory BIST Architectures"*, IEEE, Design Automation and Test in Europe Conference, 1999, pp. 872-881