A programmable BIST architecture for clusters of Multiple-Port SRAMs

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

Published in the Proceedings of the IEEE International Test Conference (ITC), 3-5 Oct. 2000, Atlantic City (NJ), USA.

DOI: 10.1109/TEST.2000.894249

A Programmable BIST Architecture for Clusters of Multiple-Port SRAMs

Alfredo BENSO, Stefano DI CARLO, Giorgio DI NATALE, Paolo PRINETTO

Politecnico di Torino
Dipartimento di Automatica e Informatica
Corso duca degli Abruzzi 24 - I-10129, Torino, Italy
Email: {benso, dicarlo, dinatale, prinetto }@polito.it
http://www.testgroup.polito.it

Monica LOBETTI BODONI
Siemens Information and Communication Networks S.p.A.
Castelletto di Settimo Milanese
I-20019 Milano MI, Italy
Email: monica.lobettibodoni@icn.siemens.it

Abstract[1]

This paper presents a 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 many distributed multi-port SRAMs of different sizes (number of bits, number of words), access protocol (asynchronous, synchronous), and timing.

1. Introduction

Multi-port SRAMs are nowadays widely used as embedded memories in a plenty of digital systems like telecommunications ASIC's or in multiprocessor systems. They allow to speed up the system, particularly when the memory has to serve many concurrent requests. Today's technologies allow the design and manufacturing of memory chips up to 11 ports, and Multi-port RAM generators are commonly available in many ASIC vendors library as LSI-Logic, Texas Instruments and ST Microelectronics. Due to the high complexity of this new integrated circuits, resorting to BIST techniques is nowadays a must. In this scenario, the test engineer has to define the BIST strategy of a complex SoC including several multi-port SRAMs of different sizes (number of bits, number of words), access protocol (asynchronous, synchronous), and timing. Apart from the required design time, the mentioned task usually poses several issues, including minimizing BIST area and routing overhead, selecting the proper number of BIST controllers to be used (that is, choosing the proper memory clustering for BIST controller sharing), fulfilling power budget constraints, and supporting diagnostic capabilities.

Commercial tools are nowadays available for the automatic insertion of the RAM BISTing [1], [2]. The present paper presents the efforts and the results obtained in designing a proprietary BIST architecture to tackle the above-mentioned set of problems.

A BIST architecture based on a μ-programmable BIST controller used to test large capacity dynamic memories was proposed by [3].

The BIST architecture proposed in this paper (Figure 1) is characterized by:

- A single *BIST Processor,* in charge of performing the test of all (or a subset of) the SRAMs of the system. It has been implemented as a micro-programmable architecture, executing elementary test primitives stored in a dedicated memory, thus capable of running any required March algorithm (it is optimized to implement the March Test for Multi Port SRAMs presented in [4]);
- A *Wrapper* around each SRAM. Each wrapper is composed of a set of *Port-Wrappers* (one per each memory port) and of a *Dispatcher.*
- Each *Port-Wrapper* contains the standard memory BIST blocks (i.e., an address generator, a background pattern generator, and a comparator), and an interface block designed to manage the

[1] This work was partially supported by the MURST (Ministero per l'Università e la Ricerca Scientifica e Tecnologica) under the project 2000 GRAAL (Generatore di celle Ram ad Altissima Affidabilità per applicazioni Life e safety-critical)

communications between the SRAM and the BIST Processor, regardless the memory access protocol.

- The Dispatcher is a simple FSM designed in order both to serially collect the test primitives for the various ports and to deliver them to the various Port-Wrappers.
- A minimal set of *Communication Signals* allowing the BIST Processor to execute and synchronize the test algorithm of all the memories under test;
- A *scan chain* connecting all the Port-Wrappers to allow full diagnosis of the memories under test.

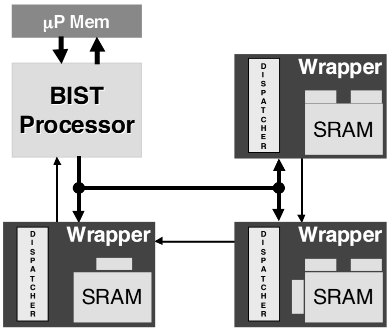

Figure 1: Basic Architecture

The proposed scheme presents several advantages. Among the others, we would like to highlight the following ones:

- It allows running concurrently the BIST of a set of SRAMs with different number of ports, sizes, accessing protocols and timing.
- The set of memories to be tested can be freely selected by the designer, using either ad-hoc test primitives stored in the test program, or a dedicated scan chain to properly set an ad-hoc status bit in each memory.
- Using a single BIST controller and a minimum set of communications signal allows minimizing the BIST area overhead and the connectivity around each SRAMs.
- Implementing the BIST Processor as a micro-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.

The paper is organized as follows: Sections 2 and 3 describe the two main blocks that compose the proposed architecture. Section 4 details the diagnostic capabilities of the architecture, whereas Section 5 presents two possible optimization to minimize the area overhead when dealing with a set of identical memories and to reduce the test length using a topological approach. Experimental results gathered on a realistic case study are discussed in Section 6, and Section 7 eventually draws some conclusions.

2. The BIST Processor

As introduced in the previous section, the proposed scheme is based on a single BIST Processor used to test all the memories of the system. To increase flexibility, the BIST execution is based on a micro-programmable approach. The *test algorithm* (a March Algorithm [4]) is stored in a dedicated *µProgram-Memory*, coded through a set of *test primitives*. The µProgram-Memory can be either a ROM or an In-System Programmable module. In the former case, the test program is fixed at design time, whereas in the latter one a custom and appropriate test algorithm can be loaded into the memory at test time.

The BIST Processor reads from the *µProgram-Memory* one test primitive at a time, forwards it to all the Wrappers of the SRAMs under test, and waits until its completion in 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 RAM. If a fault is detected, the faulty RAM can be located resorting to a set of diagnostic facilities (See Section 4).

The BIST processor and the µProgram-Memory architectures are strongly influenced by the multi-ports March Test characteristics. Due to the possibility of accessing several cells concurrently, new fault models must be used [4] and ad-hoc March Algorithms must be adopted to cover these new fault types. In particular, the proposed implementation is optimized in order to implement the March Algorithms able to cover *Complex Coupling Faults* [5]. The main characteristic of the algorithm is the access to the various ports using nested cycles:

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

where $\Uparrow^{A-1}_{B=0} \Uparrow^{n}_{C=B+1}$: denotes a nested addressing sequence, whereby the cell B goes from 0 to $A-1$; and for each value of B, cell C goes from $B+1$ to n.

Table I summarizes the set of *test primitives* needed to implement such a March Algorithm.

Test primitive	***Description***
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
ADD	Decrement the address generator
LOAD	Load a value in the address generator (see 3.2)
NME	New March Element
NOP	No Operation
NEXTBP	Next Background Pattern
CONF	Define the set of SRAM under test
END	End of test

Table I: March Algorithm Test Primitives

Each test-program step is coded in the μProgram-memory as a sequence of test primitives, one for each memory port.

As an example, let's consider the following March Algorithm used to test an 8-bit dual port SRAM (the convention for the operation is (portA:portB)):

$$\{\Uparrow(w0:w0);\Downarrow(r0:r0,w1:w1);\Uparrow(r1:r1);$$

M_0 M_1 M_2

$$\Updownarrow(w_{BP0}:w_{BP0},r_{BP0}:r_{BP0},\ldots,w_{BP7}:w_{BP7},r_{BP7}:r_{BP7});$$

M_3

$$\Updownarrow(w0:-);\ \Updownarrow_{v=0}^{n-1}(\Updownarrow_{a=0}^{v-1}(w1_a:r0_v,w0_a:r0_v,n:r0_v));$$

M_4 M_5

$$\Updownarrow_{v=0}^{n-1}(\Updownarrow_{a=v+1}^{n-1}(w1_a:r0_v,w0_a:r0_v,n:r0_v));$$

M_6

$$\Updownarrow(w1:-);\ \Updownarrow_{v=0}^{n-1}(\Updownarrow_{a=0}^{v-1}(w0_a:r1_v,w1_a:r1_v,n:r1_v));$$

M_7 M_8

$$\Updownarrow_{v=0}^{n-1}(\Updownarrow_{a=v+1}^{n-1}(w0_a:r1_v,w1_a:r1_v,n:r1_v))\}$$

M_9

The March elements M_0-M_3 realize the MATS algorithm, properly expanded as proposed in [6] to cover intra-word CFsts faults, whereby BP_0 through BP_7 are taken from the set of Background Patterns from Table II [6].

j	***Background Pattern***
0	00000000
1	11111111
2	00001111
3	11110000
4	00110011
5	11001100
6	01010101
7	10101010

Table II: 8 bits Background patterns BP_j for CFsts

The March elements M_4-M_9 represent the March 2PF2,2 proposed in [4] to test wCF_i&wCF_i.

The proposed March Algorithm can be coded using the set of primitives shown in Table III.

March Element	*Primitive*	
	Port A	*Port B*
$\Uparrow(w0:w0)$	NME	NME
	INC	INC
	W0	W0
$\Downarrow(r0:r0, w1:w1)$	NME	NME
	R0	R0
	W1	W1
	DEC	DEC
$\Uparrow(r1:r1)$	NME	NME
	R1	R1
	INC	INC
$\Updownarrow(w_{BP0}:w_{BP0}, r_{BP0}:r_{BP0}, \ldots, w_{BP7}:w_{BP7}, r_{BP7}:r_{BP7})$	NME	NME
	W0	W0
	R0	R0
	W1	W1
	R1	R1
	NEXTBP	NEXTBP
	INC	INC
$\Updownarrow(w0:-)$	NME	NOP
	W0	NOP
	INC	NOP
$\Updownarrow_{v=0}^{n-1}(\Updownarrow_{a=0}^{v-1}(w1_a : r0_v, w0_a : r0_v, n : r0_v));$	NME	NME
	W1	R0
	W0	R0
	NOP	R0
	NOP	INCCOND
	INC	NOP
$\Updownarrow_{v=0}^{n-1}(\Updownarrow_{a=v+1}^{n-1}(w0_a : r1_v, w1_a : r1_v, n : r1_v))$	NME	NOP
	NOP	LOAD
	NOP	ADD
	NOP	NME
	W0	R1
	W1	R1
	NOP	R1
	NOP	INC
	INC	COND
…..		
	END	END

Table III: March Algorithm Representation

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 [7], and an over-dissipation of power may seriously damage the devices. Moreover, the variety of SRAMs 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 special test primitive in the μProgram-Memory, as part of the test algorithm, or setting a dedicated flag into the memory Wrapper through a scan chain. Only the Wrappers of the selected memories will execute the test primitives received from the BIST Processor. In this way, several test algorithms may be storeed in the µProgram-Memory and may be applied sequentially to different sets of memories.

3. Wrapper Structure

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

The Wrapper generates the correct test patterns and memory addresses required to execute the received test primitives, and evaluates the output results of a *read-and-verify* primitive.

The Wrapper architecture consists of:

- a Dispatcher that receives from the BIST Processor the test primitives for all the ports and distributes them accordingly;
- a Port-Wrapper for each RAM port. It generates the test patterns (address and data) and verifies the correct behavior of the memory according to the command received from the dispatcher. The result of each primitive is signaled via an output line.

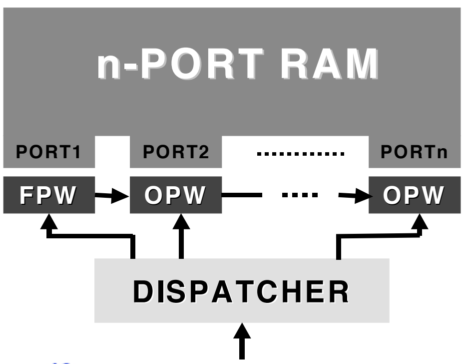

Figure 2: Wrapper architecture

Two kinds of Port Wrapper are available: one for the first port of each memory *(FPW, First Port Wrapper)* and one for the others ports *(OPW, Others Port Wrapper)*. The main difference between the two lays in the fact that each OPW receives as an input the address value generated by the previous port wrapper.

3.1. Dispatcher

The dispatcher receives the test primitives for all the port wrappers from the BIST Processor. The BIST Processor sends a test command per clock cycle to all the dispatcher (the first command is driven to all the FPWs, the second one to all the first OPWs,etc.).

Since each wrapper has no information about the other wrappers' size, a run signal is sent after all the commands. The dispatcher saves all the commands in a temporary register and, when receiving the run signal, it delivers them to each port wrapper.

As an example, the execution of the (W0:R0) instruction for a dual port memory is shown in Figure 3.

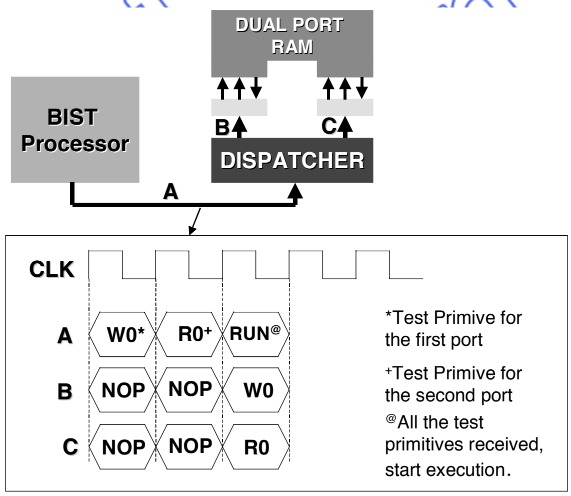

Figure 3: Test Instruction execution diagram

3.2. Port Wrapper

The internal structure of a FPW is drawn in Figure 4. The *Address Generator (AG)* is in charge of generating the correct address where the test pattern, provided by the *Background Pattern Generator (BPG)*, has to be written or verified. Several BPGs are available, to target different faults type [6]. The correctness of the content of a memory cell is evaluated through a simple C*omparator*.

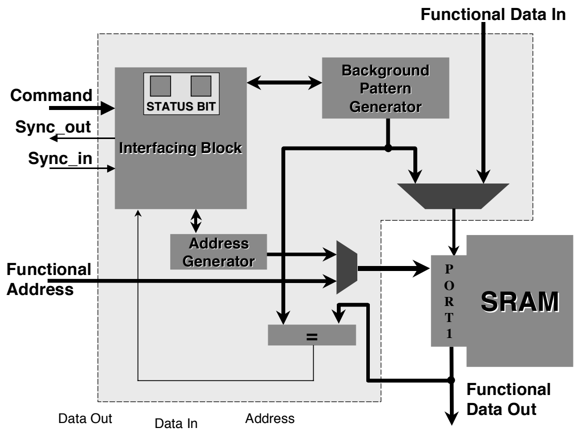

Figure 4: Wrapper structure

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 BIST algorithm completion (the *Result Status Bit*), respectively. To set and read them, the status bits of all the Wrappers are connected by two scans chain, respectively called *NormTest_Scan_Chain* and *Results_Scan_Chain*, respectively.

Finally, each FPW includes an *Interface Block* able to receive the test primitives from the Dispatcher, to receive a synchronization signal from the previous port wrapper, and to produce the *output synchronization signals* needed by the BIST Processor to schedule the next test primitive to be executed. The *output synchronization signal* assumes different meaning depending on the received test primitive (Table IV).

Received Primitive	***Output synchronization signal meaning***	***Rationale***
Write / Read	*End of Instruction (EOIN)*	Set to '1'when the instruction is finished and the input synchronization signal is equal to '1'. In this way the BIST Processor receives the logic-AND of the output signals generated by the memories under test and the input EOIN signal of the BIST Processor switches to '1' only when all the EOIN signals of the memories under test have been set to '1', i.e., all the memory Wrappers has completed the execution of the instruction
Inc/Dec CondInc/CondDec	*End of Address (EOAD)*	Set to '1' when the whole addressing space has been visited by the AG
NextBP	*End of Background Pattern (EOBP)*	Set to '1' when all the background patterns have been used
End	*Results*	Set to the logic AND among the result start bit and the synchronization signal of the preceding port wrapper
During Diagnosis	*ScanResult*	Set to the results status bits in order to form the *Results_Scan_Chain*
During scheduling configuration	*ScanSched*	Set to the mode status bits in order to form the *NormTest_Scan_Chain*

Table IV: Meanings of the Output Synchronization signal

The structure of the OPW is similar to the FPW. In order to execute the March algorithm seen in 2, this wrapper includes some additional blocks, since it must generate a subset of the entire addressing space, depending on the address generated by the previous port wrapper. An OPW is able to execute the test primitives IncCond, Dec Cond, Load, Add and Sub.

3.3. Multiplexing

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).

4. Diagnosis

When a faulty memory is detected, the proposed approach allows collecting diagnostic information concerning the location of the faulty SRAM, the ports where the fault was detected, the address of the faulty cell, and the detecting pattern. These information items are 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*. In particular, 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 5):

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

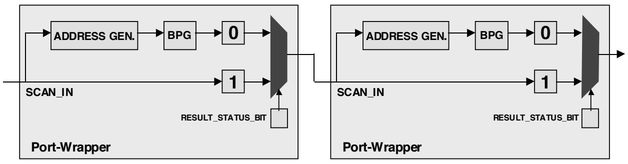

Figure 5: Results_Scan_Chain

5. Further optimizations

5.1. Sharing Wrappers among SRAMs clusters

To further reduce the BIST area overhead, the designer can share a single Wrapper for a cluster of identical SRAMs (same type, width, and addressing space).

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 RAM. In this way, when a fault is detected, the Result Status Bit of the faulty memory is set, the RAM is disconnected, and the Wrapper continues testing the remaining memories of the cluster. Obviously, in this case, the status of the Address Generator and the BPG of the faulty RAM are not preserved. To collect diagnostic information, the test must be re-executed targeting the faulty RAM, only, by properly setting its Mode Status Bit.

5.2. Using a Topological approach for complex coupling fault testing

The approach proposed in this paper is useful to describe March Algorithms for multi-port RAMs with complexity of $O(n^m)$ where n is the number of cells and m the number of ports. For practical applications, these algorithms result in very long test sequences. It is possible, as proposed in [5], to optimize the address generator of each OPW in order to generate the address for a Topological Approach. The approach consists in detecting all coupling faults between adjacent cells only. Using this optimization the test complexity can be reduced to O(n) without significant fault coverage reduction.

6. Case study

A case study has been used to evaluate the proposed approach and to gather experimental results. The circuit, named VC12AD, is a part of a telecommunication ASIC designed by Italtel SpA. The same circuit has also been used by both Italtel SpA and Siemens ICN 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 SRAMs of different sizes.

The VC12AD counts up to 860K Synopsys™ equivalent gates (excluding RAMs), plus 36 small-sized SRAMs, for a total of 14,704 bits (Figure 6).

The case study aims at evaluating:

- the BIST architecture complexity when applied to a set of SRAMs with very different characteristics;
- the area overhead after the BIST insertion.

6.1. Case Study Architecture

Figure 6 and Figure 7 show a conceptual view of the VC12AD organization and its actual floor plan, respectively. The 36 SRAMs of the circuit are grouped in four distinct macro-areas whose characteristics are listed below.

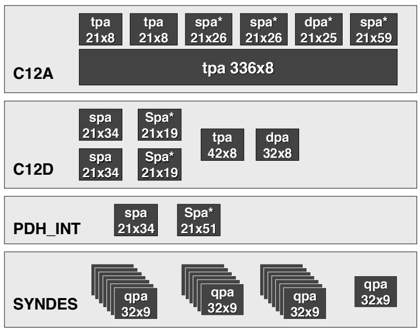

Figure 6: VC12AD memories organization

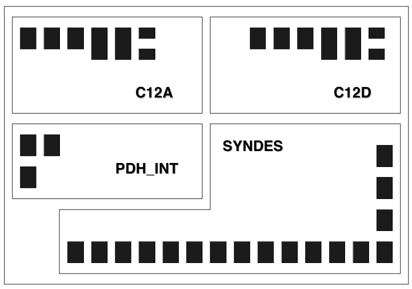

Figure 7: VC12AD floorplan

- C12A : it contains 7 RAMs:

n. of instances	Type[2]	Size[3]
2	tpa	21x8
2	spa*	21x26
1	dpa	21x25
1	spa	21x59
1	tpa	336x8

- C12D : it contains 6 RAMs:

n. of instances	Type[1]	Size[2]
1	dpa	32x8
1	tpa	42x8
2	spa	21x34
2	spa*	21x19

- PDH INT : it contains 2 RAMs:

n. of instances	Type[1]	Size[2]
1	spa	21x34
1	spa	21x51

- **SYNDES** : It consists of 21 identical blocks. Each of them contains one instance of a qda 32x9 (asynchronous quadruple port RAM with two ports dedicated to write and two dedicated to read).

6.2. Case Study BIST Architecture

In the BIST Architecture definition, we tried to minimize the number of wrappers resorting, whenever possible, to clusters of SRAMs (see Section 5.1). 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 as four clusters of 7, 7, 6, and 1 element, respectively.

The design of the BIST architecture has been strongly influenced by the actual floor plan, where, for example, the 3 spa21x34 SRAMs (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 8.

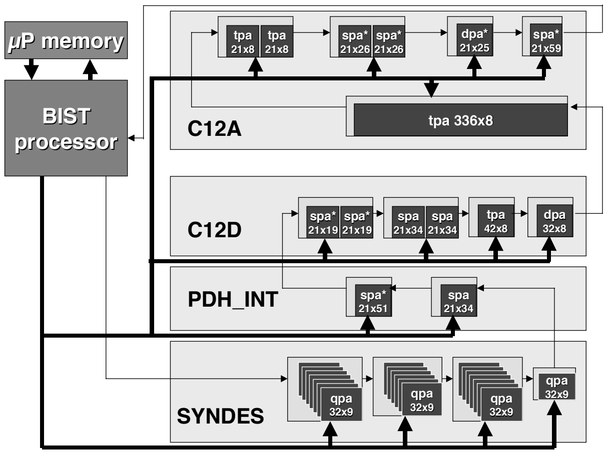

Figure 8: VC12AD BIST Architecture

6.3. Case Study BIST Scheduling

Due to the different characteristics of the VC12AD SRAMs (read/write ports, read-only ports, and write-only ports are present), it is not possible to adopt a unique March Algorithm for the overall circuit. We have thus been forced to organize the BIST in four session, each one using an appropriate 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.

6.4. Experimental results

The area occupation of each memory and its Wrapper is in Table V, whereas Figure 9 shows the contributions of the functional blocks of each Wrapper.

---

2 spa: single port asynchronous RAM;
spa*: single port asynchronous RAM with 1 write enable for each data bit;
dpa : dual port asynchronous RAM (one port dedicated to write and one dedicated to read);
tpa : triple port asynchronous RAM (one port dedicated to write and two ports dedicated to read);

3 (Number of words) x (bits per word)

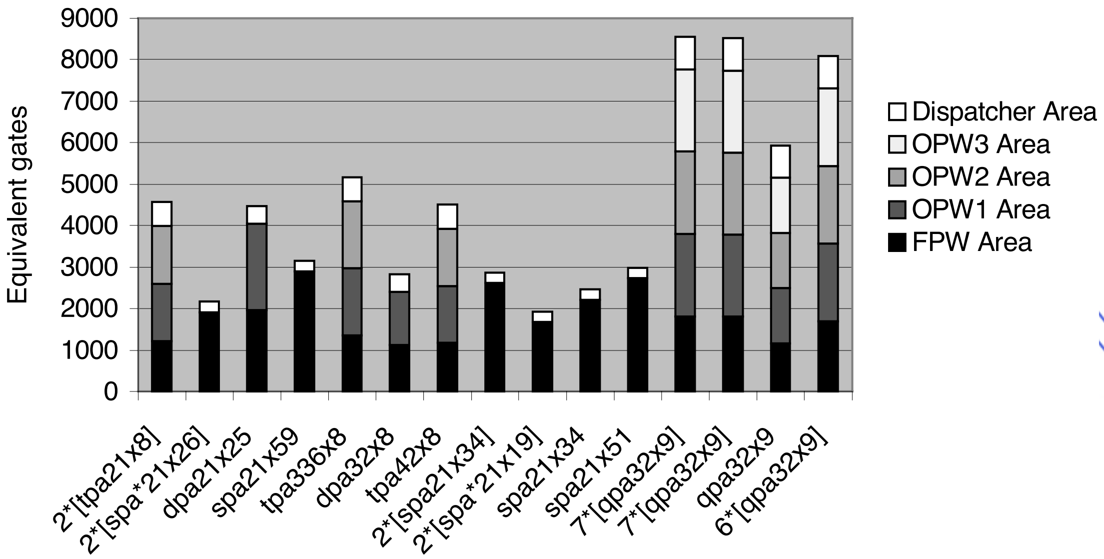

Figure 9: Wrappers area

RAM	Wrapper Area
2*[tpa21x8]	4,574
2*[spa*21x26]	2,165
dpa21x25	4,470
spa21x59	3,148
tpa336x8	5,169
dpa32x8	2,829
tpa42x8	4,509
2*[spa21x34]	2,870
2*[spa*21x19]	1,925
spa21x34	2,162
spa21x51	2,989
7*[qpa32x9]	8,543
7*[qpa32x9]	8,543
6*[qpa32x9]	8,084
qpa32x9	5,924

Table V: Memory Wrapper overhead

The total area overhead including the Wrappers and the BIST Processor is in Table VI.

Glue Logic area	862,347
Total RAM area	380,503
Total Wrapper area	68,177
BIST processor area	5,431
μProgram memory area	4,459
Total	1,320,917
Total area overhead	**6,28%**

Table VI: Total area overhead

As shown in Table VI, the BIST processor and the μProgram-memory area overhead is a fix contribution and it is not influenced by the number of SRAMs in the system.

6.5. Comparison with a commercial tool

To evaluate its effectiveness, we compared the area overhead introduced by the proposed approach with the one obtained using a commercial tool on the same test case. The area overhead introduced by the tool is around the 8%, and therefore slightly higher than the one obtained inserting the proposed BIST schemes. Nevertheless, it is necessary to take into account that the mentioned test case has been specifically chosen to stress the tool and, probably, on a real system the overhead would be smaller. Moreover, our approach is designed to target memories only, whereas the commercial tool is able to introduce test logic for all the different parts of the circuit.

7. Conclusions

In the present paper we presented a proprietary solution for a particular industrial scenario, in which it is necessary to define the BIST strategy of a complex system including several multi-port SRAMs of different sizes, access protocol, and timing. The proposed architecture consists in a single BIST Processor, implemented as a micro-programmable machine and able to execute different test algorithms, a Wrapper for each SRAM (or SARAM cluster), each Wrapper including one *Port-Wrapper* for each memory port and a special block named *Dispatcher*.

Each *Port-Wrapper* instiantes standard memory BIST modules, and an interface block to manage the communications between the SRAM 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. To begin with, it allows running concurrently the BIST of a set of SRAMs of different number of ports, sizes, accessing protocols and minimizing the BIST area overhead and connectivity around each SRAMs. 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, ROMs, legacy cores, etc) have to be BISTed as well, more complex and sophisticated approaches (such as the HD^2BIST architecture [8] [9]) have to be adopted.

8. References

[1] Logic Vision web site, http://www.logicvision.com, February 2000

[2] Mentor Graphics web site, http://www.mentrog.com/dft, February 2000

[3] [KTTa90] Hiroki Koike, Toshio Takeshima, Masahide Takada, *A BIST scheme using microprogram ROM for large capacity memories*, IEEE International Test Conference (ITC'90), pp 815 – 822

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

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

[6] A.J. van de Goor, I.B.S. Tlili, *March tests for word-oriented memories*, DATE'98: IEEE Design, Automation and Test in Europe, pp. 501-508, 1998

[7] Y. Zorian, *A distributed BIST Control Scheme for complex VLSI devices*, VTS'93: The 11th IEEE VLSI Test Symposium, pp. 4-9, April 1993

[8] A. Benso, S. Chiusano, S. Di Carlo, P. Prinetto, F. Ricciato, M. Spadari, Y. Zorian *HD^2BIST: a Hierarchical Framework for BIST Scheduling, Data patterns delivering and diagnosis in SoCs*, Submitted to IEEE International Test Conference (ITC00), Atlantic City (NJ), USA, October 2000

[9] A. Benso, S. Cataldo, S. Chiusano, P. Prinetto, Y. Zorian, *HD-BIST: a Hierarchical Framework for BIST Scheduling and Diagnosis in SoCs*, IEEE International Test Conference (ITC'99), Atlantic City (NJ), September 1999, pp. 993-1000

[10] M. Franklin, K.K. Saluya, *Built-in Self-Testing of Random-Access Memories*, IEEE Computer, October 1990, pp.45-56

[11] LSI Logic web site,http://www.lsil.com/, February 2000