Politecnico di Torino

Porto Institutional Repository

[Proceeding] An effective distributed BIST architecture for RAMs

(Article begins on next page)
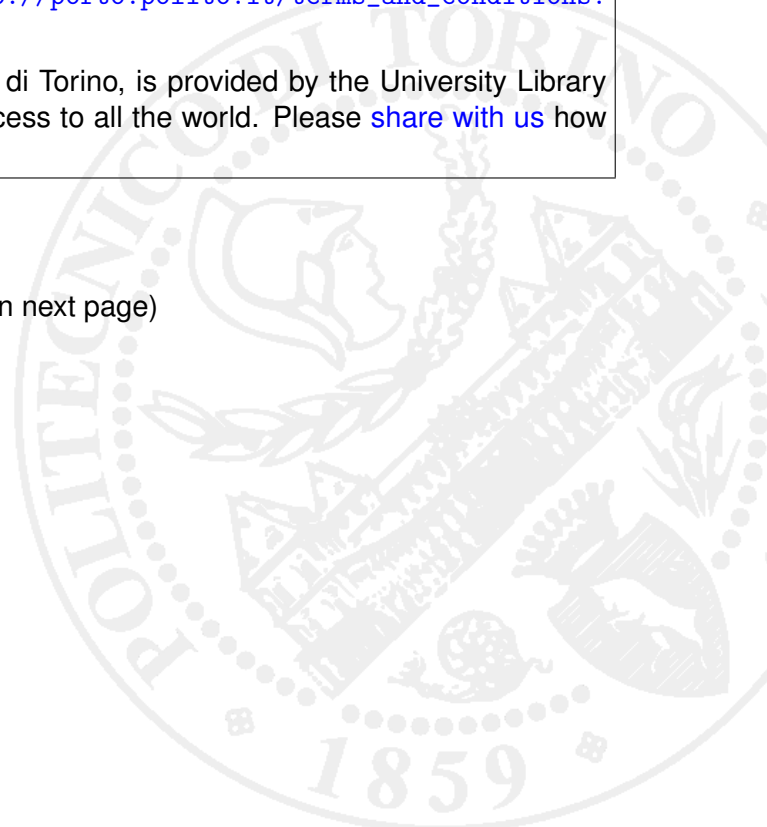
An Effective Distributed BIST Architecture for RAMs

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

Alfredo BENSO, Silvia CHIUSANO, 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, chiusano, dicarlo, dinatale,
prinetto}@polito.it*

Abstract

The present paper proposes a solution to the problem of testing a system containing many distributed memories of different sizes. The proposed solution relies in the development of a BIST architecture characterized by a single BIST Processor, implemented as a micro-programmable machine and able to execute different test algorithms, a Wrapper for each SRAM including standard memory BIST modules, and an interface block to manage the communications between the SRAM and the BIST Processor. Both area overhead and routing costs are minimized, and a scan-based approach allows full diagnostic capabilities of the faults possibly detected in the memories under test.

1. Introduction

Several commercial tools are nowadays available for the automatic insertion of the RAM BISTing [1], [2]. This paper presents the efforts and the results obtained in designing a proprietary BIST architecture to fulfill a peculiar industrial scenario.

In the target industrial scenario, the test engineer has to define the BIST strategy of a complex System-on-Chip including several 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 many issues, as the BIST area and routing overhead, the number of BIST controller to be used, the power budget constraints, and the diagnostic capabilities of the approach.

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

- A single *BIST Processor*, able to perform the test of all (or a subset of) the SRAMS of the system. It is implemented as a micro-programmable machine executing elementary test primitives stored in a dedicated memory and implementing any required March algorithm;
- A *Wrapper* placed around each SRAM, including standard memory BIST blocks (i.e., an address generator, a background pattern generator, and a comparator), and an interface block designed to manage the communications between the SRAM and the BIST Processor independently from the memory access protocol;
- A minimal set of *communication signals* that allows the BIST Processor to execute and synchronize the test algorithm of all the memories under test;
- A scan chain connecting all the Wrappers in order to allow full diagnosis of the memories under test.

The proposed scheme presents several advantages. To begin with, it allows running concurrently the BIST of a set of SRAMs of different sizes, accessing protocols and timing. Moreover, the set of memories to be tested can be flexibly defined by the user, using either ad-hoc test primitives stored in the test program, or a dedicated scan chain configuring a status bit in each memory. The use of a single BIST controller and a minimum set of communications signal allow minimizing the BIST area overhead and the routing around each SRAMs. Finally, implementing the BIST Processor as a micro-programmable machine provides the test engineer with 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.
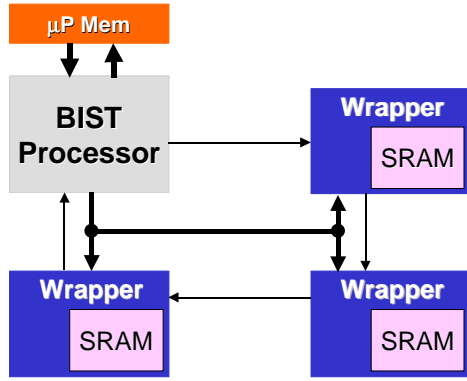
Figure 1: Basic Architecture

The paper is organized as follows: Sections 2 and 3 describe the two main blocks that compose the proposed approach. Section 4 details the diagnostic capabilities of the architecture, whereas Section 5 presents a possible optimization when dealing with a set of identical memories. 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 [3]) is stored in a dedicated **μ**Program-Memory, coded using a set of *test primitives*. The μProgram-Memory can be either a ROM (in this way the test program is fixed at project time) or a programmable memory (in this way the appropriate test algorithm can be loaded into the memory at test time). The BIST Processor reads one test primitive at a time, forwards it to all the Wrappers of the SRAMs under test using a synchronization signal, and waits for all the enabled SRAMs to complete the test primitive before sending the next one. When the test program is completed (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 set of test primitives needed to code a March Algorithm is listed in Table 1.

Table 1: March Algorithm Test Primitives

Test primitive	Description
CONF	Define the set of SRAMs under Test
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
DEC	Decrement the address generator
NEXTBP	Next Background Pattern
END	End of test

As an example, let's consider the MATS algorithm for an 8-bit wide RAM, properly expanded as proposed in [4] to cover intra-word CFsts faults:

$$\{\Uparrow(w0) \; ; \; \Downarrow(r0,w1) \; ; \; \Uparrow(r1) \; ;$$
$$\Updownarrow(w_{BP0}, r_{BP0}, w_{BP1}, r_{BP1}, ..., w_{BP7}, r_{BP7}),)\}$$

whereby BP0 through BP7 are taken from the set of Background Patterns from Table 2 [4].

Table 2: 8 bits Background patterns BP$_j$ for CFsts

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

The considered MATS algorithm can be described using the following sequence of primitives:

Table 3: Modified MATS Algorithm

March Element	*Primitive*
$\Uparrow(w0)$	W0
	INC
$\Downarrow(r0,w1)$	R0
	W1
	DEC
$\Uparrow(r1)$	R1
	INC
$\Updownarrow(w_{BP0}, r_{BP0}, w_{BP1}, r_{BP1}, ..., w_{BP7}, r_{BP7},)$	W0
	R0
	W1
	R1
	NEXTBP
	INC
---	END

An important issue to be faced when running concurrently the BIST of many modules is fulfilling power budget constraints. In fact, BIST typically results in

a circuit activation rate higher than the normal one [5], and an over-dissipation of power may seriously damage the device. Moreover, the wide 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 placed under test 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 it is possible to store in the μProgram-Memory more than one test algorithm and apply them to different sets of memories. The two scheduling mechanisms are briefly explained in the following two subsection.

2.1. Scheduling using the *"CONF"* primitive.

Using the CONF primitive, it is possible to embed into the test Program the scheduling information. The format of this primitive in the μProgram-Memory is shown in Table 4.

Table 4: Conf primitive representation

CONF
#words
ActivationMask

Where:
- *Conf* is the primitive opcode;
- *#words* is the number of 4-bit words used to code the *ActivationMask*;
- *ActivationMask* is a mask of bits, one for each 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 consider the system in Figure 2:
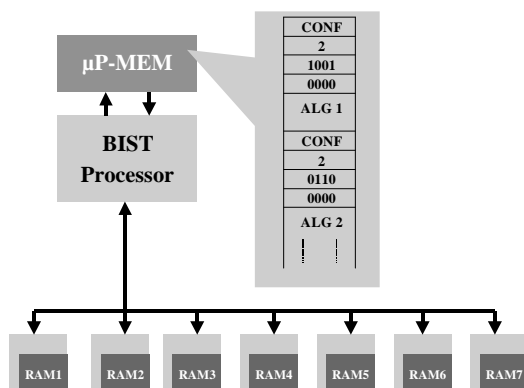
Figure 2: Scheduling using the "Conf" primitive.

When the BIST processor reach a CONF primitives during the Test Program execution it read the ActivationMask and configure all the memory wrappers using the scan chain defined in Section 1 in order to realize the described scheduling plane. The first ActivationMask described in Figure 2 sets the RAM1 and RAM4 under test whereas the second one sets the 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 stop the test program execution and wait for a new start command to continue with the next one.

2.2. Scheduling using the Scan chain option.

In order to give high flexibility to the designer, the set of RAMs under test can be set loading the appropriate *ActivationMask* (see 2.1) directly from the extern using a scan chain protocol.

In order to choose the appropriate test algorithm in the μ-program memory, also the *μ-program memory Address Register* can be loaded via scan chain protocol.

3. The memory Wrappers

The Wrapper placed around each memory has to execute the test primitives received by the BIST Processor, independently of the memory access protocol. Moreover, the Wrapper is the only element in the architecture that must know the dimension 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 primitive, and evaluates the output results of a *read-and-verify* primitive.

The internal structure of a Wrapper is in Figure 3. 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. The BPG can generate "1…1" and "0…0" test patterns as well as the background patterns shown in Table 2. The correctness of the content of a memory cell is evaluated using a simple *comparator*.

Two Status Bits are used 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*), respectively. In order to load and read their content, the status bits of all the Wrappers are connected by two different scans chain, named *Normal_Test_Scan_Chain* (NTScan) and *Results_Scan_Chain* (Resscan).

Finally, each Wrapper includes an *Interface Block* able to receive the test primitives from the BIST Processor, and to produce the status signals needed by the BIST Processor to schedule the next test primitive to be

executed. In particular, the Interface Block generates the following information:

- End of Instruction (EOIN): asserted when the last received test primitive is thoroughly executed;
- End of Address Space (EOAD): asserted when the address generator reaches the end of its addressing space;
- End of Patterns (EOPG): asserted when the BPG has generated the whole set of background patterns;
- Read-and-Verify Result (GO): asserted when the content of the addressed memory cell matches the value expected by the test algorithm.
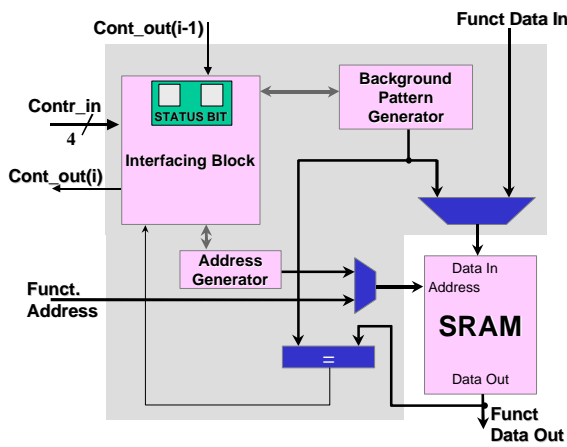
Figure 3: Wrapper structure

The BIST Processor receives the logic-AND of the signals generated by the memories under test. In this way, for example, the input EOAD signal of the BIST Processor switches to '1' only when all the EOAD signals of the memories under test have been set to '1', i.e., all the memory Wrappers reached the end of their address space. Consequently, from the BIST Processor point of view, the system under test consists in a single memory, whose size is equal to the maximum size of the memories under test. 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 (Figure 4), and all the information items routed using only five signals (four command signals and one synchronization signal).
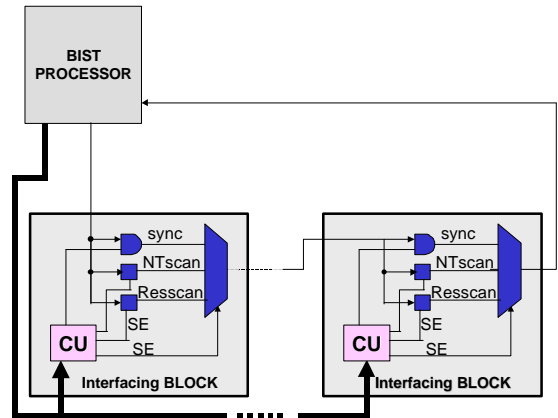
Figure 4: Multiplexing of command and synchronization signals

4. Diagnosis

When a faulty memory is detected, the proposed approach allows collecting diagnostic information concerning the location of the faulty SRAM, the address of the faulty cell and the pattern that detected the fault. These information are stored into the Result Status Bit, the Address Generator, and the Background Pattern Generator of each Wrapper. All the diagnostic information can thus be accessed via the *Results_Scan_Chain*. In particular, depending on the result of the test, each Wrapper is able to configure its portion of the *Results_Scan_Chain* in one of the following two ways (Figure 5):

- If the RAM is not faulty, only the Result_Status_Bit (whose value is equal to '0') is placed on the scan chain.
- If a RAM is faulty, the Result_Status_Bit (whose value is '1') is concatenated to the contents of the Address Generator and the Background Pattern Generator.
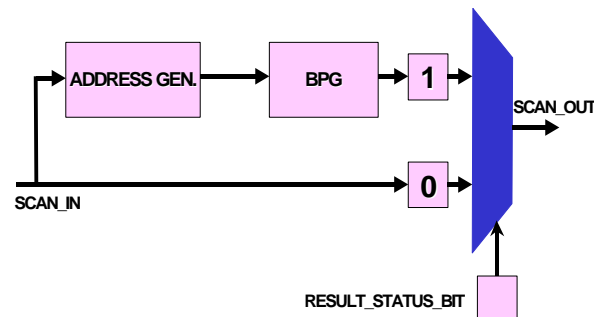
Figure 5: Results_Scan_Chain

5. Further optimizations

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). When the BIST Processor drives the Wrapper, only one Address Generator and one BPG are needed to execute the required test primitives on all the SRAMs.

The only difference with the previously described Wrapper structure is that a shared Wrapper contains a pair of Status Bits and a comparator for each RAM (Figure 6).

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, properly setting its Mode Status Bit.
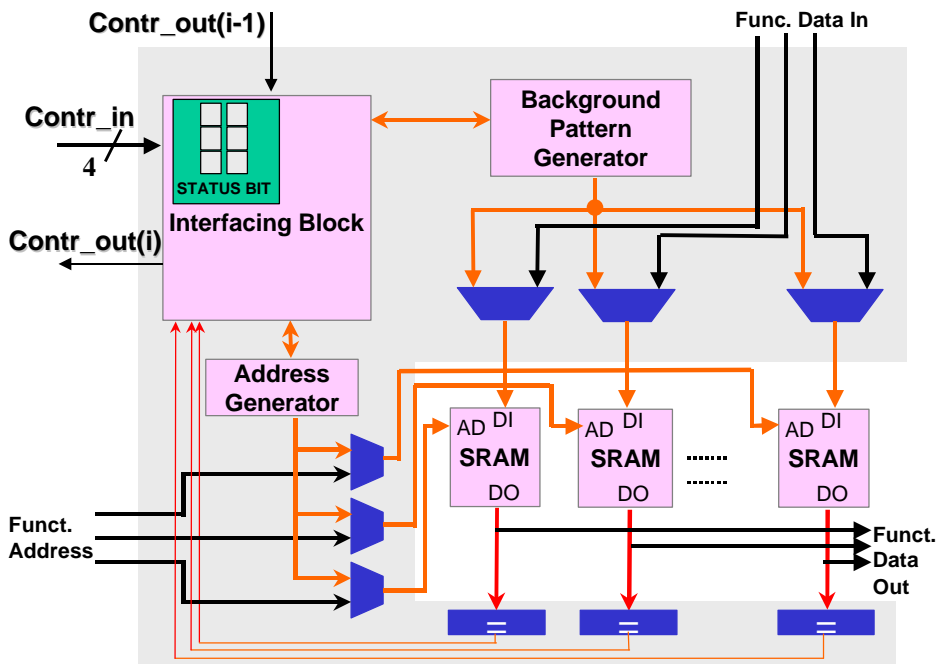

Figure 6: Wrapper structure for a RAM cluster

6. Experimental results

To evaluate the impact of the proposed solution in terms of area overhead, a case study has been developed within Siemens ICN. The target circuit (Figure 8) has been described in VHDL and synthesized using the *G10 LSILogic library* [6], which provides a set of SRAMs of different sizes. The test case includes 8 SRAMs: 5 different memories managed by 5 different Wrappers and a cluster of 3 identical memories that shares a single Wrapper.

The area occupation of each memory and its Wrapper is in Table 5 whereas Figure 8 shows the contributions of the functional blocks of each Wrapper.

The total area overhead including the Wrappers and the BIST Processor is in Table 6 and Figure 9.
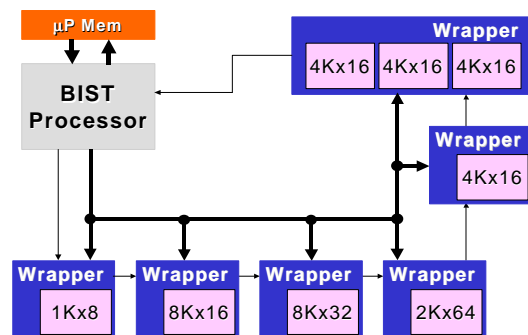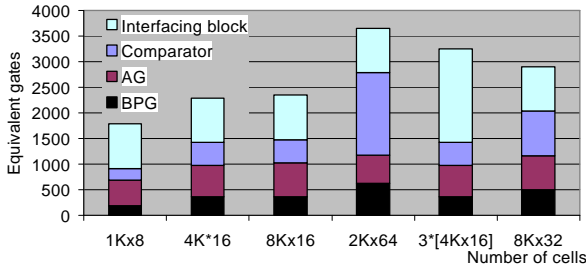

Figure 7: Case study
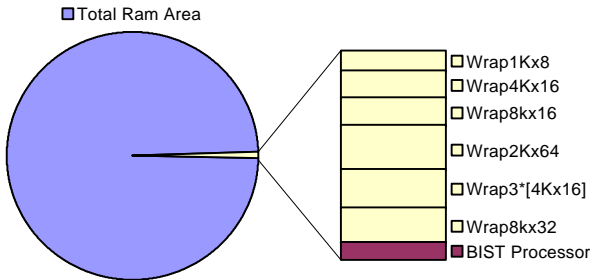
Table 5: Memory Wrapper overhead

RAM	RAM Area	Wrapper Area	Overhead
1Kx8	25,831	1,788	6.92%
4Kx16	139,740	2,291	1.64%
8Kx16	265,355	2,347	0.88%
2Kx64	314,262	3,653	1.16%
3*[4Kx16]	419,220	3,254	0.78%
8Kx32	492,537	2,908	0.59%

Figure 8: Wrappers area

Table 6: Total area overhead

|	
Total RAM area	1,656,945
Total Wrapper area	16,241
BISTprocessor area	1,392
Total	1,674,578
Total area overhead	**1.06%**

Figure 9: Area overhead

Resorting to the experimental results shown in the previous tables, we can relate the area overhead of the Address Generator and the BPG to the dimension of the memory under test.

Figure 10 and Figure 11 present the trend of the area occupation of the mentioned two blocks related to the number of bits and number of words, respectively.
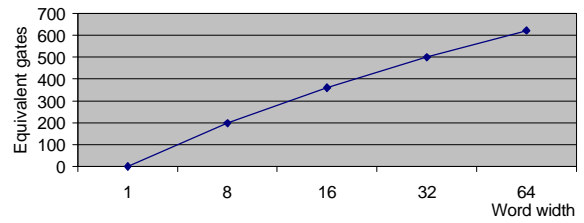
Figure 10: BPG area evaluation
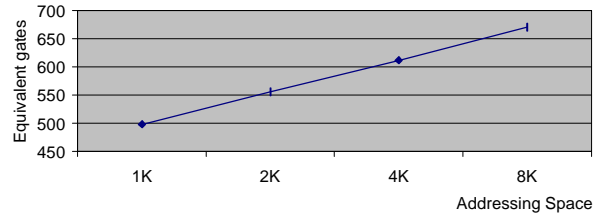
Figure 11: Address generator area evaluation

7. Conclusions

In this paper we presented a possible solution to a particular industrial scenario, in which it is necessary to define the BIST strategy of a complex system including several SRAMs of different sizes, access protocol, and timing. The proposed architecture is composed of a single BIST Processor, implemented as a micro-programmable machine and able to execute different test algorithms, a Wrapper for each SRAM including standard memory BIST modules, and an interface block to manage the communications between the SRAM and the BIST Processor. The proposed scheme presents several advantages. To begin with, it allows running concurrently the BIST of a set of SRAMs of different sizes, accessing protocols and timing minimizing the BIST area overhead and the routing around each SRAMs. Moreover, 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.

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] A.J. Van de Goor, *Using March Tests to Test SRAMs*, in IEEE Design and Test, March 1993, pp 8-14

[4] 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

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

[6] http://www.lsil.com