A Programmable Logic BIST Controller for IP Cores

Alexandre M. Amory[1], *Fernando Moraes*[2], *Marcelo Lubaszewski*[3]

[1] *Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS)*

[2] *Instituto de Informática – Pontifícia Universidade Católica do Rio Grande do Sul(PUCRS)*

[3] *Dep. de Engenharia Elétrica – Universidade Federal do Rio Grande do Sul (UFRGS)*
Porto Alegre - Brazil - E-mail: luba@eletro.ufrgs.br

Abstract

This paper proposes a flexible and reusable BIST controller for the test of logic IP cores. Such a controller, called ProBIST, provides test programmability and flexibility for the BIST architecture, making it easier the reuse of the IP core and the test structure. The ProBIST processor can be programmed according to the core user's requirements. In this way, the proposed BIST architecture can be created by the core vendor according to the core's implementation details, offering to the core user an additional degree of flexibility, i.e. the choice of an efficient BIST solution. The integration of the ProBIST controller into a P1500 wrapper, also presented in this paper, provides a standard test interface between the BISTed core and the final system. The proposed strategy is discussed considering area and performance experimental data.

Keywords – programmable test, test controller, logic BIST, core-based test, P1500, LFSR reseeding.

1. Introduction

The increasing system complexity combined with the shortened life cycles for the electronic products leads the manufacturers to adopt new design methodologies. One of the solutions is the reuse of IP blocks. The core users can purchase functional blocks from IP vendors and integrate them using user-defined logic to build the SoC.

The increased complexity and reduced time-to-market of the systems also impact the test methodology. The complete SoC has to be tested for manufacturing faults considering reduced accessibility to the cores. Several standards and techniques emerged to reduce the effort for testing those systems.

Using traditional test approaches, the core vendor can only provide a minimal test structure for the core, since most of the test decisions are task of the system integrators. Traditionally, adding BIST to a hard core could reduce its test flexibility since the core user could not modify it. However, often only the core provider is familiar with the core's implementation details. Thus, if the core provider does not include BIST in the core, it is hard for the system designer to integrate this block efficiently since the intellectual property (IP) is usually protected.

The use of *programmable* BIST has been presented in the literature as an interesting approach to increase the flexibility of BIST structures. Programmable BIST assumes the integration of test-specific processors to the system. The main advantages and drawbacks of programmable BIST are:

- The *programmability* and *flexibility* of the test, since modifications in the test can be implemented by simply reprogramming the test processor, without changing the hardware design;
- It allows *at-speed testing* in reduced *test tim*e due to its specialized nature;
- Most of the test control task is embedded into the processor's control block. External access is necessary only to load a very small test program, to active the processor, and to read the results. Therefore, a low cost tester can be used;
- The drawback of this approach is a small *area overhead* as demonstrated in this paper.

This paper describes the ProBIST, a test processor that implements a single and flexible BIST module, so that the same core can be easily integrated and tested in a number of different systems, without hardware redesign. With such a flexible test scheme, one can ensure the reuse of test modules in addition to the reuse of the functional block. Programmability and flexibility of the test structure is achieved without significant penalties in the test performance and area overhead at the system level. With the proposed approach, core vendors can insert a flexible and programmable BIST logic even in hard IP cores. The system integrators can configure the test parameters according to the system requirements without any redesign of the BIST logic. Adding BIST to the cores also reduces the number of patterns stored in the ATE, which leverages the external tester requirements and cost. The main *contribution* of this work is to demonstrate the advantages of a test processor integrated to P1500-compliant IP cores rather than a hardwired BIST logic.

The paper is organized as follows. Section 2 reviews some related works. Section 3 describes the proposed BIST test controller. Section 4 discusses the integration of ProBIST into a P1500 wrapper. Section 5 presents some experimental results. Section 6 concludes the work.

2. Related Work

Several papers have demonstrated the advantages of using BIST schemes based on multiple polynomials and LFSR reseeding [3][4][5][6]. However, there is a lack of flexible and reusable hardware to support those

BIST schemes.

Hellebrand *et. al.* propose in [3] a BIST scheme that supports LFSR with multiple polynomials and seeds. The seeds are obtained by solving systems of linear equations. There is a programmable BIST scheme to run the LFSR. However, this hardware needs a custom decoding logic for each core under test (CUT) to encode the polynomials. Thus, reuse of the BIST architecture is not addressed.

Fagot *et. al.* [4] propose a fast simulation based method to compute an efficient seed of a LFSR. The method is intended to produce a one-seed test sequence of a given test length aiming to concentrating on the hardest to detect faults of the CUT and to achieves a high fault coverage.

Krishna and Touba [6] propose a loss-less test vector compression technique which combines LFSR reseeding and statistical coding to combine a high encoding efficiency.

In [7], an embedded test controller is proposed to test some blocks of the SoC that require at-speed testing. Such a controller, called MET, is embedded into the system with the purpose of test data manipulation and controlling. As the test controller is synthesized in the same technology of the SoC, at-speed test can be performed. The MET approach relies on the definition of specific instructions. Additionally, the test vectors and response are assumed to be stored in the memories that are usually available in the SoC.

Some recent works [8][9][10] propose the integration of programmable BIST specific to memory cores. The BIST processor can be programmed with different memory test algorithms such as March algorithms.

In this paper, the idea of making the BIST more flexible is also explored, but in a more general way. First, generic logic cores are considered, rather than memory blocks. Second, LFSR reseeding is systematized through the definition of a single controller that implements the functions of reusable LFSR and MISR. Finally, the integration of this controller to a P1500 wrapper allows the system integrator to access and re-program the BIST logic using standard communication protocols, thus easing the core test.

3. ProBIST Test Controller

This section presents the proposed architecture for the flexible, at-speed and low cost BIST test controller.

3.1. ProBIST – Programmable BIST Processor for Logic Cores

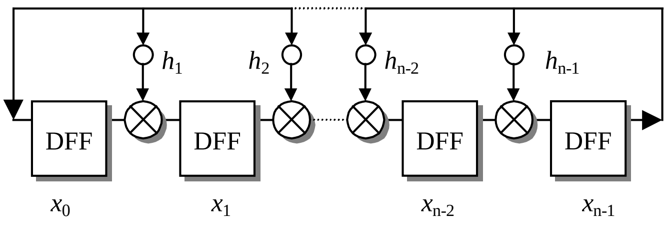
ProBIST is a programmable BIST controller that can be used to generate pseudo-random test patterns and compact test responses. It has been specially developed to be used in the context of system-on-chip designs, as an embedded test source and/or sink. The test pattern/response generator/compactor can be completely and easily configured using the defined instruction set. For example, ProBIST can be used to test cores using LFSR with different polynomials and seeds, increasing the pseudo-random test efficiency [3]. Additionally, ProBIST can be programmed to generate/compact test patterns/responses cycle by cycle, i.e. enable at-speed testing.

The programmable test controller is described in a configurable VHDL description. The core vendor can easily configure the BIST structure using the two integer generic variables that parameterize the data registers and the BIST counter. The data register is configured according to the target IP core data width, while the BIST counters are adjusted according to the maximal number of patterns required.

3.2. Generic BIST Logic

The generic BIST logic is the main module of ProBIST. It has a pattern generator and a signature analyzer. Among the several BIST modules available in the literature, the modular LFSR and MISR [1] were implemented in the ProBIST processor, since they can be easily configured to use different combinations of seeds and polynomials.

Figure 1 illustrates a canonical modular LFSR. Intuitively, one can observe that the bit flip of the LFSR occurs only when the value of the last LFSR stage is a logic 1.



n represents the number of LFSR stages. h_i represents polynomial coefficient, where $0 \le i \le n$-1. When h_i has value 1 there is a feedback in the coefficient *i*.

Figure 1 – Canonical modular LFSR example [1].

$$\begin{bmatrix} X_0(t+1) \\ X_1(t+1) \\ X_2(t+1) \\ \mathrm{M} \\ X_{n-3}(t+1) \\ X_{n-2}(t+1) \\ X_{n-1}(t+1) \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & \Lambda & 0 & 0 & 1 \\ 1 & 0 & 0 & \Lambda & 0 & 0 & h_1 \\ 0 & 1 & 0 & \Lambda & 0 & 0 & h_2 \\ \mathrm{M} & \mathrm{M} & \mathrm{M} & & \mathrm{M} & \mathrm{M} & \mathrm{M} \\ 0 & 0 & 0 & \Lambda & 0 & 0 & h_{n-3} \\ 0 & 0 & 0 & \Lambda & 1 & 0 & h_{n-2} \\ 0 & 0 & 0 & \Lambda & 0 & 1 & h_{n-1} \end{bmatrix} \begin{bmatrix} X_0(t) \\ X_1(t) \\ X_2(t) \\ \mathrm{M} \\ X_{n-3}(t) \\ X_{n-2}(t) \\ X_{n-1}(t) \end{bmatrix}$$

Figure 2 – Equation of a generic modular LFSR [1].

Bardell [1] argued that the equivalent mathematical implementation of the canonical modular LFSR presented in Figure 1 is the equation presented in Figure 2. Besides, an equivalent implementation can be expressed by the code presented in Figure 3, which has been inferred from Figure 1. These two equivalent implementations of the modular LFSR, Figure 2 and Figure 3, have been compared in Matlab™ environment for several different BIST configurations. In each iteration the resulting value of both implementations were compared.

The variables *polynomial* and *state* of Figure 3 are equivalent to h_i and X_i(t) of Figure 2. Implementing a shift right and incrementing the data when the MSB (Most Significant Bit) is 1 is equivalent to a rotate instruction. The current value of LFSR (*state*) is *xored* with the *polynomial* when the state MSB is one.

```
void lfsr( unsigned polynomial, int *state)
{  int i = *state;
   i <<=1;
   if (*state < 0){ // if MSB = '1'
        i++;
        i ^= polynomial;
   }
   *state = i;
}
```

Figure 3 – Generic modular LFSR subroutine.

From the implementation of the modular LFSR presented in Figure 3, it was possible to infer the generic hardware implementation of the modular LFSR, as shown in Figure 4. There is one register to store the polynomial value, one register for the current value of the LFSR, a rotate right (RR) logic, *xor* gates and two multiplexers. When the value of the MSB of the LFSR is the logic 1, the next value of the LFSR will be a rotate of the current LFSR value plus a *xor* with the polynomial register. When the value is 0, the next value of the LFSR will be only rotated.

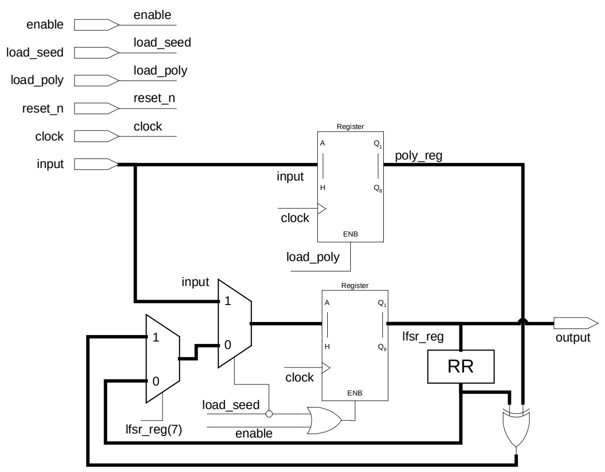


Figure 4 – The generic LFSR module configured as 8bit.

The generic MISR module can be implemented following the same method presented for the LFSR.

Using the generic BIST logic shown in Figure 4, it is possible to use the same module with different BIST configurations without any hardware redesign, increasing the reuse of the test structure. In fact, the ProBIST controller is a generic LFSR, a generic MISR, and an additional simple control logic responsible for loading the polynomials and seeds. This simple hardware design is the most important characteristic of ProBIST to achieve low area overhead and at-speed test.

The area overhead of a generic LFSR like the one presented in Figure 4 is typically smaller than three times a LFSR with fixed polynomial with minimal number of feedback coefficients. On the other hand, the generic LFSR is configurable and the same module can be used in different test configurations, whereas with fixed LFSRs, it would be necessary one LFSR per scenario.

3.3. Instruction Set

Table 1 presents the ProBIST instruction set. Before running the BIST logic, the processor must be configured by defining the polynomial and seed. The instructions that configure the LFSR and MISR are *ld_lfsr_seed*, *ld_lfsr_poly*, *ld_misr_seed* and *ld_misr_poly*. Since the modules are configured, they can be executed for *n* cycles with the instructions *run_lfsr*, *run_misr* and *run_lfsr_misr*. At the end of the test, the processor can perform two types of evaluation: just send through the output port the generated signature (*rd_sig*) or to compare the generated signature against the expected one (*cmp_sig*). The *halt* instructions indicate the end of the test.

The processor starts to operate after the reset event. The sequences of operations are: (*i*) read an instruction; (*ii*) decode the instruction; (*iii*) read the data field (operand) if applied; (*iv*) execute the operation.

Table 1 – The BIST processor instruction set.

Instruction	Description
LD_LFSR_SEED	set the seed of the LFSR
LD_LFSR_POLY	set the polynomial of the LFSR
LD_MISR_SEED	set the seed of the MISR
LD_MISR_POLY	set the polynomial of the MISR
RUN_MISR	run the MISR for *n* cycles
RUN_LFSR	run the LFSR for *n* cycles
RUN_LFSR_MISR	run the lfsr and the misr for *n* cycles
RD_SIG *	read the generated signature
CMP_SIG	compare the generated signature against the expected one
HALT *	stop the processor

* indicates no operands associated with the instruction

An important characteristic of the instructions set is the absence of loops. Thus, the ProBIST does not need test memory to store the instructions. As it will be detailed in Section 4.1, the instructions are loaded serially. Approaches like [8] use test memory to load the instructions, which increases the area overhead. The test memory is necessary since there are loops in the instruction set, *i.e.* instructions are executed several times, while in ProBIST instructions are executed only once. The fact that the instructions in ProBIST are loaded serially does not limit the at-speed test, because when the execution instructions (*run_lfsr*, *run_misr* and *run_lfsr_misr*) are running they can generate patterns/analyze responses cycle by cycle.

4. Test Architecture

This section presents the proposed test architecture, which combines a programmable test processor, and a P1500 compliant test wrapper. Figure 5 shows this test architecture. It is important to highlight that the integration of ProBIST and P1500 is only one example of application. ProBIST could be integrated to others test architectures as well as P1500.

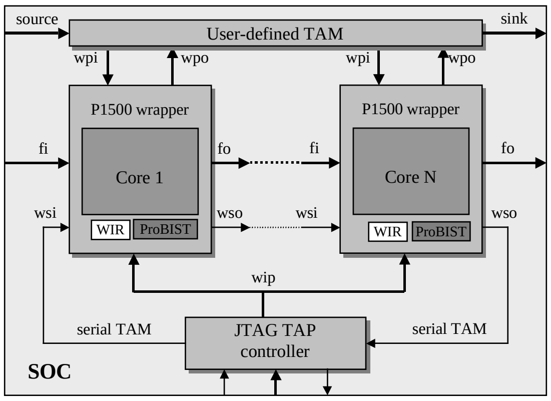


Figure 5 – The test architecture including ProBIST.

One can observe that the test architecture is almost the same originally presented by Marinissen *et. al.* in [2]. The difference is the ProBIST processor embedded into the wrapper and the additional test access port (TAP) instructions necessary to interface the ATE with the test controller.

4.1. Test Wrapper

Figure 6(a) presents the ProBIST controller connected to a core and wrapped using the P1500 test wrapper. Figure 6(b) and Figure 6(c) present the wrapper boundary IO cells. It is possible to observe in Figure 6(b) that the input cell includes an additional multiplexer to enter the test pattern generated by ProBIST.

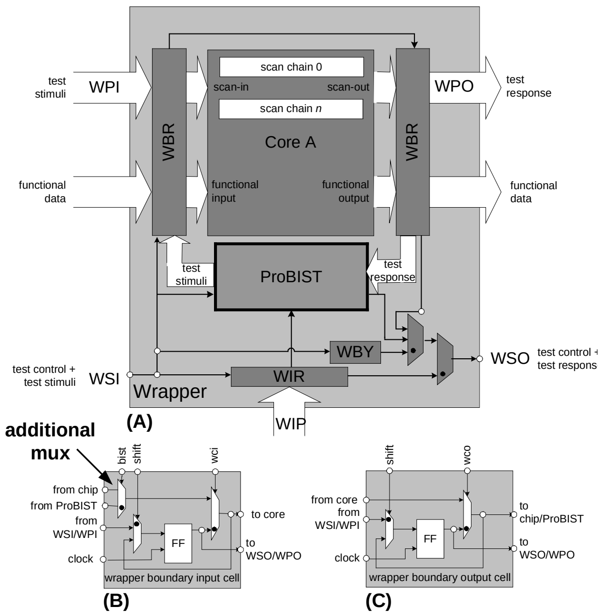


Figure 6 - P1500 test wrapper with ProBIST. (a) wrapper, (b) input cell and (c) output cell.

Figure 7 presents the ProBIST module. The data and instruction buffers, represented by *operandReg* and *opcodeReg*, are serially loaded, through the signal WSI, with the operand and opcode data, respectively. This structure avoids the inclusion of a test memory to the embedded controller, reducing the area overhead. ProBIST can be easily configured like full scan design as illustrated. In self-test mode the *operandReg* and *opcodeReg* register are concatenated with ProBIST internal registers.

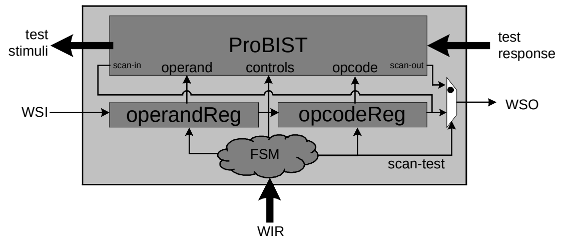


Figure 7 – Detail of the ProBIST.

4.2. New Instructions to TAP Controller

Besides the mandatory IEEE 1149.1 instructions, the TAP controller should support the following instructions:

- *ResetProBIST*: initialize ProBIST registers;
- *ShiftInstruction*: shift ProBIST instructions to the *operandReg* and *opcodeReg* buffers;
- *LoadInstruction*: Load the *operandReg* and *opcodeReg* buffers into the ProBIST internal register;
- *ReadSignature*: load the signature to the *operandReg* buffer before shifting it through WSO.

4.3. Typical Usage of ProBIST

In the typical scenario where the ProBIST controller is used, ProBIST instructions are stored in the ATE memory. The instruction opcodes are 4 bit wide, while the operands width depends on the CUT test interface. For example, if the CUT has a 16-bit test interface, each memory word must have 20 bits. The number of memory words necessary to store the test program depends on the test configuration, but it typically needs 7 words as demonstrated in Table 2. The size of the test program increases linearly with the number of seeds and/or polynomials required.

Table 2 – Minimal test program example.

	Mnemonic	Operand
1	LD_LFSR_SEED	x"A5A5"
2	LD_LFSR_POLY	x"002C"
3	LD_MISR_SEED	x"A55A"
4	LD_MISR_POLY	x"0038"
5	RUN_LFSR_MISR	x"0400"
6	CMP_SIG	x"ABCD"
7	HALT	-

Table 2 presents a minimal ProBIST test program. The instructions 1 to 4 configure the seed and polynomial of the LFSR and MISR. The seed can be any number between 0 and 2^{16}-1. The instruction

LD_LFSR_POLY loads the polynomial x"002C, which represents the primitive polynomial $x^{16} + x^5 + x^3 + x^2 + 1$. One can observe that the absence of coefficient in the polynomial is represented by a bit '0', otherwise is represented by bit '1', excepted by the least and most significant coefficients. The MISR polynomial, represented by x"0038", is $x^{16} + x^5 + x^4 + x^3 + 1$. The instruction 5 configures ProBIST to run in parallel the LFSR and the MISR for 1024 patterns/responses. After the pattern generation/compactation the resulting signature in the MISR is compared with the value x"ABCD". If it does not match the error flag is activated. An alternative to instruction 6 is to execute instruction RD_SIG that sends the MISR contents to the *operandReg* register to be evaluated a posteriori.

Table 3 – Test program example with LFSR reseeding.

	Mnemonic	Operand
1	LD_LFSR_SEED	x"A5A5"
2	LD_LFSR_POLY	x"002C"
3	LD_MISR_SEED	x"A55A"
4	LD_MISR_POLY	x"0038"
5	RUN_LFSR_MISR	x"0400"
6	**LD_LFSR_SEED**	**x"DCBA"**
7	**RUN_LFSR_MISR**	**x"0100"**
8	CMP_SIG	x"4321"
9	HALT	-

The example test program presented in Table 3 performs LFSR reseeding in instruction 6 and 7. The instruction 6 loads a new seed in the LFSR and the instruction 7 runs the LFSR and MISR for more 256 cycles.

The test code presented so far comprises only ProBIST instruction set, but in a tester, ProBIST instruction must be interleaved with IEEE 1149.1 instructions presented in Section 4.2. A typical tester memory contents is presented in Table 4. Italic instructions represent the IEEE 1149.1 instructions.

Table 4 – Typical tester memory contents necessary to activate ProBIST.

```
ResetProBIST
ShiftInstruction
LD_LFSR_SEED x"A5A5"
LoadInstruction
ShiftInstruction
LD_LFSR_POLY x"002C"
LoadInstruction
...
ShiftInstruction
CMP_SIG x"4321"
LoadInstruction
ReadSignature
ShiftInstruction
HALT
LoadInstruction
```

The combination of the ProBIST controller and the P1500 infrastructure increases the flexibility of the test solution with minimal area overhead as demonstrated in Section 5. An IP core can be configured to run different pseudo-random configurations considering the system requirements, *i.e.* more or less number of patterns, test time or LFSR seeds/polynomials. An interesting test configuration to pseudo-random resistant cores is to combine pseudo-random and external test patterns to reduce the number of patterns stored in an expensive ATE [3]. This can be easily integrated using the propose architecture.

5. Experimental Results

The ProBIST controller was described in VHDL and synthesized with Leonardo Spectrum™ synthesis tool to evaluate area consumption and performance, having as target device a 0.35 micron TSMC ASICs. The synthesis was executed with typical values and low effort.

Three others general propose processors cores have been synthesized for comparison reasons: FemtoJava is a academic processor which interprets Java byte-codes (available at http://www.inf.ufrgs.br/sashimi); 8051 is an 8051-like 8-bit micro-controller (available at http://www.cs.ucr.edu/~dalton/i8051/i8051syn); MIPS is a commercial processor with load-store architecture and pipeline with three stages. It supports interrupts and all MIPS-I user mode instructions (available at http://www.opencores.org/projects/mips).

In [8], a hardware overhead of about 800 gates has been presented for the memory test controller. However, this approach needs a test memory of size 43x4bits to store the test program, which is not included in this overhead. Including the test memory overhead, the total overhead would be about 3000 gates (800 for the test processor and 2200 gates for the test RAM). This extra overhead has been calculated considering a VHDL memory description synthesized for a 0.35 μm library. It is also important to highlight that the [8] approach is specific to memory cores rather than a generic logic BIST approach presented in this paper.

As presented before, ProBIST can be configured by two constants, which parameterize the data registers and the BIST counter. Table 5 presents area and performance with different values for those constants.

Table 5 – Comparative performance and area for the proposed approach.

cores	area (eq. gates)	frequency (MHz)
RAM BIST [8] approach	800+2200	40
FemtoJava (16 bits)	4023	67
8051 (8 bits)	6512	98
MIPS (32 bits)	19243	57
ProBIST - 16 bits data regs 10 bits counter	1291	198
ProBIST - 16 bits data regs 16 bits counter	1432	154
ProBIST – 32 bits data regs 10 bits counter	2313	194

It is not part of this work to evaluate fault coverage, since it heavily depends on the seeds, polynomial and number of hard-to-detect faults of the CUT. Any of the reseeding techniques presented in Section 2 can be used to improve the fault coverage of the CUT.

6. Conclusions

This paper presented a programmable logic BIST test controller integrated to a P1500 compliant wrapper. The main advantages explored in this paper are: total reuse of the ProBIST considering different contexts, test programmability according to the system requirements, full support to multiple seeds/ polynomials, reduction of the number of patterns in the ATE, at-speed testing, very low area overhead compared to similar approaches, configurable VHDL description. The drawback of the presented approach is that the small area overhead can either be significant to small cores like the FemtoJava processor.

The main contributions of this work are: (*i*) to the best of our knowledge, the design of new and totally reusable BIST modules based on modular LFSR; (*ii*) the demonstration of advantages of a programmable logic BIST processor combined with the standardized P1500 test wrapper.

It is expected that the proposed wrapper design can motivate core vendors to integrate BIST solutions in the logic IP cores, in addition to memory IP cores.

Acknowledgements - The authors acknowledge Érika Cota to the helpful suggestions during this work.

7. References

[1] Bardell, P.H. “Built-In Test for VLSI: Pseudorandom Techniques”. John Wiley & Sons, 1987, 354 p.

[2] Marinissen, E.J.; Kapur, R.; Lousberg, M.; McLaurin, T.; Ricchetti, M. and Zorian, Y., "On IEEE P1500's Standard for Embedded Core Test," in *Journal of Electronic Testing: Theory and Applications*, vol. 18, 2002, pp. 365-383.

[3] Hellebrand, S.; Wunderlich, H.J. and Hertwig, A. "Mixed-Mode BIST Using Embedded Processors". IEEE International Test Conference, 1996, pp. 195-204.

[4] Fagot, C.; Gascuel, O.; Girard, P. and Landrault, C. "On Calculating Efficient LFSR Seeds for Built-In Self-Test". IEEE European Test Workshop, 1999, pp. 7-14.

[5] Kagaris, D. and Tragoudas, S., "LFSR Characteristic Polynomials for Pseudo-Exhaustive TPG with Low Number of Seeds," in *Journal of Electronic Testing: Theory and Applications*, vol. 19-3, 2003, pp. 233-244.

[6] Krishna, C.V. and Touba, N.A. "Reducing Test Data Volume Using LFSR Reseeding with Seed Compression". IEEE International Test Conference, 2002.

[7] Cota, E.F.; Brisolara, L.; Carro, L.; Susin, A.A. and Lubaszewski, M. "MET: A Microprocessor for Embedded Test". IEEE Test of Core-Bared Systems, 2001, pp. 100-107.

[8] Appello, D.; et. al. "Exploiting Programmable BIST for the Diagnosis of Embedded Memory Cores". IEEE International Test Conference, 2003, pp. 379-385.

[9] C.-T. Huang, J.-R. Huang, C.-F. Wu, C.-W. Wu, T.-Y. Chang. “A Programmable BIST Core for Embedded DRAM”. *IEEE Design and Test of Computers*, vol. 16-1,1999, pp. 59-70.

[10] Dreibelbis, J.; Barth, J.; Kalter, H. and Kho, R. "Processor-Based Built-In Self-Test for Embedded DRAM". *IEEE Journal of Solid-State Circuits*, vol. 33-11, 1998, pp. 1731-1740.