An novel Methodology for Reducing SoC Test Data Volume on FPGA-based Testers

P. Bernardi, M. Sonza Reorda

Dipartimento di automatica e Informatica, Politecnico di Torino, IT

ABSTRACT

Low-Cost test methodologies for Systems-on-Chip are increasingly popular. They dictate which features have to be included on-chip and which test procedures have to be adopted in order to guarantee high test quality, while minimizing application costs. Consequently, Low-Cost test strategies can be run on testers offering lower performance and/or reduced features with respect to traditional Automatic Test Equipments (ATEs); these equipments are usually referred to as Low-Cost testers.

This paper proposes a methodology for reducing the test data volume for the application of SoC Low-Cost test procedures. The method exploits a tester architecture organization suitable for SoCs testing, which includes a programmable device: the usage of this configurable block joined to the analysis of test pattern regularities permits minimizing the test data volume, thus improving the tester capabilities. The proposed method relies on test pattern compression at system level and it does not address core level pattern manipulation, as several other previously published works do.

Case studies are proposed, which provide data about the application of the proposed methodology to the test of SoCs including self-testable processor and memory cores. IEEE 1149.1 and IEEE 1500 test access mechanisms are considered. The achieved pattern depth reduction ratio is up to about the 64% for the considered case studies.

1. Introduction

Low-Cost is a widely used term in the testing scenario. This attribute merges many test concepts and it is perceived with slightly different meanings depending on the considered testing area. Concerning test generation, a Low-Cost methodology provides test procedures that minimize the test application costs, such as test duration; conversely, in the test equipments scenario, a Low-Cost tester is a "cheap" equipment offering lower performance and/or reduced features with respect to a traditional Automatic Test Equipment (ATE).

With the advent of Systems-on-Chip (SoCs), the term Low-Cost is commonly used to classify a set of *strategies* and *equipments* that exploit Design-for-Testability (DfT) features included on-chip for reducing test costs without impacting on test effectiveness; the costs of SoC test procedure involves many factors, that are primarily the test pins count, the required application frequency and the amount of patterns to be physically applied. Therefore, if test resources are judiciously partitioned, testers would be asked to provide only few channels per chip, the frequency requirement can be relaxed and even small memories can be sufficient to store the pattern set.

Low-Cost properties of a SoC test procedure normally stem from two separate test aspects:

- Test initialization/activation/management and result retrieval via test access protocols
- Test execution.

The adoption of test access protocols to transport information inside the SoC architecture mainly addresses pin count reduction, often at the expense of the bandwidth. Indeed, autonomous test procedure execution addresses frequency requirements mitigation, since it normally exploits internal or independent clock supply resources that do not request any external intervention.

Concerning test data volume of procedures usually classified as Low-Cost, some additional considerations have to be done; the overall number of test vectors to be applied to the DUT depends on the number of initialization and management operations required to activate the SoC test functionalities. Consequently, patterns describing such procedures finally reside on the tester memory and potentially impact on the test applicability. Moreover, the adoption of a test access protocol may heavily affect the tester memory requirements by encapsulating each core vector into longer sequences that depend on the SoC test infrastructure organization. In this paper, the attention is mainly focused on this point.

This paper describes a suitable methodology for test data volume reduction for the application of SoCs Low-Cost test procedures. The illustrated strategy relies on the capabilities of a tester equipment based on the usage of a programmable logic device and exploits a pattern depth minimization method taking advantage of test pattern regularities that derive from the specific test access mechanism employed. In the described methodology, a test set is firstly analyzed in order to identify test segments recurring several times during stimuli application; in this way, the programmable device is exploited for supplying the identified recurrent test parts without explicitly requesting their multiple memorization.

The paper presents some conceptual issues that allow test size reduction and the guidelines for an effective usage of the programmable resources considered available. A case study is discussed including the application of Software-based Self-Test to a processor core and the application of a Built-in Self-Test strategy to memory cores. IEEE 1500 and JTAG test protocols are considered as test access mechanisms.

Section 2 of the paper provides backgrounds on Low-Cost test procedure; section 3 contains key concepts for pattern size mitigation and describes a possible low-cost tester hw/sw organization, section 4 shows some obtained results, and section 5 draws some conclusions.

978-3-9810801-3-1/DATE08 © 2008 EDAA

2. Background

As mentioned in the introduction, the Low-Cost attribute is assigned to a SoC testing method dependently on both the implemented core test types and the adopted on-chip DfT solutions. These two aspects are normally strictly correlated in a strict manner. Anyway, it is important to separate what concerns the application of the test procedure (i.e., how the test is executed) from what concerns the test management at the SoC level (i.e., in which way the test is prepared, and run, and the results retrieved).

Let's briefly introduce which types of SoC test strategies are currently labeled as Low-Cost in the literature. To be as much general as possible, testing methodologies at core level may be classified in *Scan-based* and *Self-test* approaches.

Low-Cost scan-based test approaches rely on design techniques allowing the minimization of the number of tester channels [1] and tester frequency requirements [2]. In addition to traditional scan cells, such techniques adopt suitable DfT features such as decoders and PLL-based circuitries; these techniques permit moving deterministic patterns in the chip at reduced speed, thus applying them at higher frequency independently on ATE capabilities.

Low-Cost self-test approaches may be based on Infrastructure Intellectual Properties (IIPs) modules [3] or may employ functional parts of the device under test itself [4]. The key point is that, once launched, a self-test procedure is autonomously applied until it ends. A further sub-classification for Low-Cost Self-Test approaches may includes *Software-Based Self-Test* (SBST) [4] and *Built-in Self-Test* (BIST) strategies [5][6]. Test procedures exploiting both SBST and BIST principles consists at least in three parts: 1) a preliminary initialization phase aimed at loading at low frequency test micro-codes or setting parameters, 2) a self-test execution at high frequency, 3) result download at low frequency.

When merging many cores into a SoC, a test infrastructure is required to move test data from a test source to each core and test results from each core to a test sink [9]. Standardized test structures have been proposed in many papers (i.e., Virtual Socket Interface Alliance (VSIA) [7] and the IEEE 1500 embedded core test [8]). Under the hardware point of view, possible test infrastructures include wrappers and test access mechanism (TAMs); concerning software, test creation for the entire SoC consists in test translation from core terminals to SoC pins [12].

The adoption of such a standardized test structure is a key point for transforming a SoC test procedure into a low-cost one, especially with respect to the frequency and edge-sets requirements. The adoption of a suitable wrapper efficiently permits to separate the test management from the test execution clock domains. [8][11].

Unfortunately, the usage of such test structures increases the patterns size; in fact, additional sequences are required for core selection and for addressing test wrapper internal resources [12].

3. Proposed method

This paper describes a method for significantly reducing the test data volume required by SoC testing. Differently from previous works [9] which were based on test data compression and scheduling optimization, the proposed methodology is based on the identification of recurrent test set parts; recurring segments are mainly due to the Test Access Mechanism included on-chip and to the communication protocol introduced at the SoC integration level.

The purpose of the proposed approach is to reduce the test data volume size by asking the tester to intervene during the pattern application; in particular, it is requested to autonomously generates part of the test patterns that will be no longer requested to reside in the tester memory. The proposed methodology is based on

- test set analysis
- suitable Low-Cost tester organization.

The test set analysis aims at reducing the pattern set size by identifying test segments occurring several times in the considered test set. This phase requires the knowledge of the employed test access mechanism and its inputs are the whole test set description and a set of shorter test segments provided by the test engineer who developed the test recipe. The analysis process returns

- the list of test segments in the selected short test segments and identified as hardly recurrent,
- a modified test set description pruned from such recurrent test segments,
- the information needed to rebuild the original test set characteristics in a suitable format.

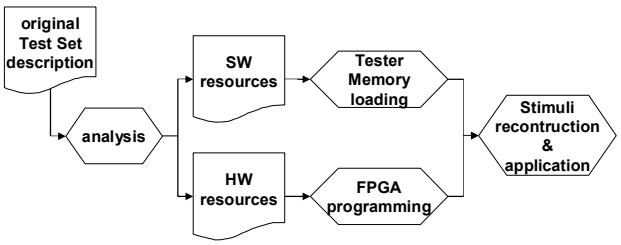

Fig. 1: Methodology flow.

The tester organization assumes that the used tester includes a programmable logic device (FPGA) devoted to reconstruct the original test set. This component is often included in tester architectures; in the proposed method it is used as it follows: first of all, identified recurrent test segments are translated into finite state machines (FSMs) able to autonomously reproduce the recurrent test segments themselves. Secondly, FSMs are mapped on the FPGA and activated at pre-calculated times during test pattern application; Additionally, the programmable logic device is asked to store suitable circuitry able to merge recurrent and non-recurrent test parts

To summarize, the overall Low-cost test analysis and application flow consists in the following, sequentially performed steps, as illustrated in Fig 1:

a. Test pattern analysis and recurrent segment identification

b. Tester FPGA programming and reduced pattern file storage on tester memory
c. Physical application of the test set autonomously reconstructed by the tester.

3.1 Recurrent test segments identification

Let's consider a test procedure suitable for Low-Cost test of SoCs. Independently on the implemented communication protocol, for every test data sent to any core or read from it, it is possible to identify three separated phases:

1. preparation of the involved test structure(s)
2. data transfer
3. return to idle state.

These three phases can be identified over the timing diagram snapshot reported in Fig. 2, which is related to the test procedure for a SoC design driven through a TAP controller compliant with the IEEE 1149.1 standard. Such a TAP controller is used to control the functionalities of IEEE 1500 wrappers [9] and its state machine is driven by 5 suitable top level signals [10].

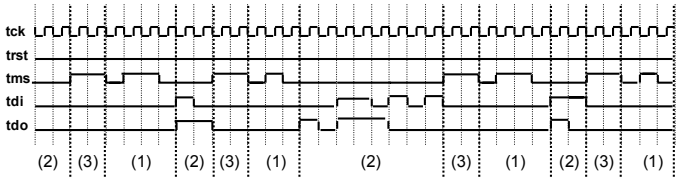

Fig. 2: TAP access timing snapshot; zone labels are related to the described phases.

Depending on the content of zones labeled as (1), the TAP controller IR or the wrapper registers are addressed. Therefore, their content is filled up accordingly with the `tdi` values serially shifted in. In the example, the TAP IR is 2 bit wise (the three possible values correspond to Wrapper Instruction Register selection-write, Wrapper Data Register write, Wrapper Data Register read operations), while the wrapper register addressed is 8 bit long.

Accordingly to these considerations, 2 types of occurrences may be identified in the pattern set:

- *Vertical occurrences*: a vertical occurrence is encountered when a timing diagram slice (during more than 1 clock cycle) is repeated many times in the overall pattern set application. Fig. 3 shows an example; 2 vertical occurrences are observable matching with zones (1) and (3) identified in fig. 2.

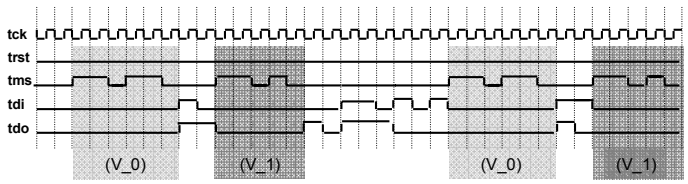

Fig. 3: Two vertical occurrences (V_0 and V_1) are identified in the proposed scenario.

- *Horizontal occurrences*: a horizontal occurrence is encountered when a signal value is maintained stable at a certain value for more than 1 clock cycle. Horizontal occurrence identification follows the vertical one, thus it works on pattern set regions that are not vertically recurrent. An example is shown in fig. 4; horizontal occurrences are shaded parts with outlined border.

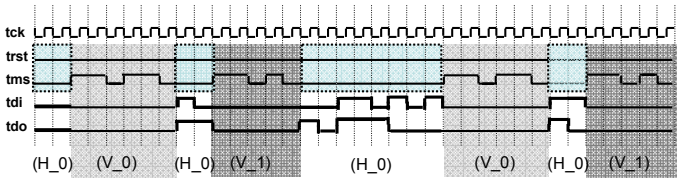

Fig. 4: One horizontal occurrence (H_0) is identified in the proposed scenario for the trst and tms signals.

By following such an occurrence identification strategy, a certain number of bits can be removed from the pattern set for being generated by hardware tester resources. The result of this pruning operation is therefore a *Reduced Test Set description file* (RTS file); clock cycle per clock cycle, this file sequentially stores the remaining bits in a fixed order. Such bits corresponds to narrower pattern parts in Fig. 4

Indeed, a second file called *Test Segments Occurrence table file* (TSO file) is required, storing the information required to reconstruct the original pattern set. The following encoding has been chosen for the TSO file:

- In case of the current pattern segment does not present any vertical occurrence,
  - 16 bits (2 bytes) are used to describe its characteristics (values are stored in the RTS file)
  - the MSB of the first byte is set to 0
  - the 2^{nd} to 4^{th} bits provide the horizontal occurrence identification number (up to 7 cases - 111 if none)
  - the remaining 12 bits store the length in clock cycle of the segment (up to 4096 clock cycles)
- In case the current pattern segment corresponds to a vertical occurrences
  - 8 bits are used to describe it in the TSO file and its value is generated by tester hardware parts
  - the MSB of this byte is set to 1
  - the remaining 7 bits indicate the vertical occurrence identification number (up to 128).

In general, the reduced test data volume (RTDV) obtained by applying the illustrated method can be calculated as:

$$RTDV = 8 \cdot (n_{TS} + n_{NRS}) + \sum_{(i=0 \text{ to } nNRS)} (l_{NRSi} \cdot (n_S - n_{HRSi}))$$

where n_{TS} is the total number of segments (either showing vertical occurrence or not), n_{NRS} is the number of segments that do not show any vertical occurrence, l_{NRSi} is the length in clock cycles of the i^{th} non-recurring segment, n_S is the number of stimulated top-level signals and n_{HRSi} is the number of signals horizontally occurring during the i^{th} non-recurring segment.

In the shown example, the RTS file (concerning the reported test set slice) would be:

```
000011100100111101100110
```

The resulting TSO file is the following:

```
00000000
00000010
10000000
00000000
00000010
10000001
00000000
00001000
10000000
00000000
00000010
10000001
```

Concerning the shown example that encompasses very few clock cycles, the RTS plus TSO size is 120 bits, while initially the test set size included 144 bits, corresponding to a test data volume reduction ratio of the 16.7%.

The effectiveness of this strategy may depend on the analyzed patterns and it fits well with test sets showing

- long vertical occurrences
- few, but extensively repeated cases of horizontal occurrences.

This is the case of Low-Cost test strategies including BIST and SBST, which usually request many repeated initialization and result download operations.

3.2 Low-Cost Tester HW/SW organization

The principle of the proposed method is that some parts of the test set can be removed from the pattern set and reproduced by means of hardware FSMs, which are activated at predetermined times like macros for programming languages.

To suit with the proposed method, the assumed Low-Cost tester architecture has to possibly include the hardware components itemized herein:

a. A *Control Unit* in charge of managing the test flow execution
b. A *Stimuli Generator* directly sequencing patterns to the device under test
c. A *Monitoring Unit* managing results retrieval
d. A *set of memories* storing test set information.

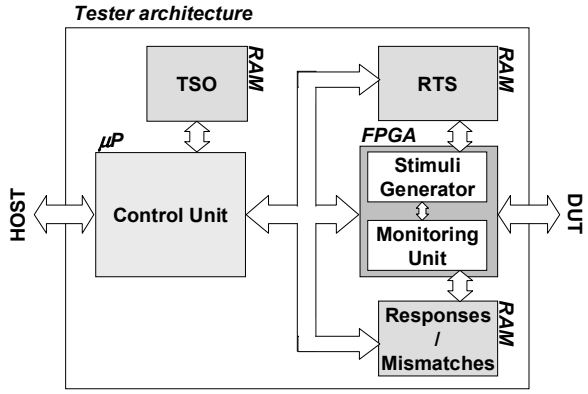

Fig. 5: hardware organization of the Low-Cost tester.

As shown in fig. 5, Control Unit tasks may be performed by a microprocessor (alternatively, it may correspond to a properly designed circuitry) Within its duties, the Control Unit reads the TSO file and communicates information to Stimuli Generator.

The Stimuli Generator is implemented by a FPGA and, by means of the information received from the Control Unit, it is in charge of reconstruct the test set by reading the RTS file and by activating some FSMs that autonomously reproduce the recurrent test segments identified during the test set analysis.

The Monitor Unit is also contained in the FPGA and it can act in two ways: pattern matching and results storage. In case of pattern matching, it simply compares the obtained DUT outputs with the expected ones; therefore, it strictly collaborates with the Stimuli Generator and memorizes only noticed discrepancies. Otherwise, when selecting results storage mode, the DUT output values are completely stored.

Following the description of such tester components, it is clear that the Memories are requested to:

- Store the TSO file
- Store the RTS file
- Store test stimuli application responses/pattern mismatches.

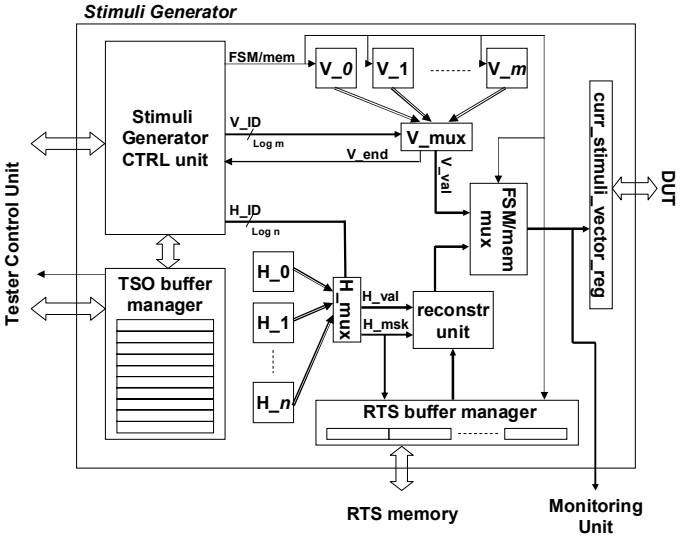

Fig. 6: Tester Stimuli Generator conceptual schema.

The most important component of the illustrated environment is the Stimuli Generator, which interacts with all the other mentioned tester components and whose conceptual schema is reported in Fig 6.

This component contains the following modules:

- A Stimuli Generator CTRL unit
- A TSO buffer and its management unit
- A RTS buffer and its management unit
- A set of V_*i* units, each one containing a FSM able to reproduce one of the identified vertical occurrences
- A set of H_*i* units, each one containing information about one of the identified horizontal occurrences
- A Reconstr Unit that merges RTS with horizontal occurrence data

- A set of multiplexers for pattern source selection
- A Current Stimuli Vector register.

The Stimuli Generator CTRL unit is in charge of both communicating with the tester control unit (merely test procedure management, e.g., test launch, status polling and conclusion acknowledgement) and managing the test stimuli reconstruction/application. The latter aspect consists in translating the TSO content in suitable signals:

- FSM/mem signal: this signal enables the application of patterns deriving from RTS file content or from FSMs reproducing vertical occurrences
- V_ID signal: identifies the specific vertical occurrence to be applied
- H_ID signal: identifies the specific vertical occurrence to be applied.

FSM/mem signal timings derives from both the TSO file content (it includes the duration in clock cycle for each non-recurrent test part) and the internal Stimuli Generator CTRL unit. In fact, other than reproducing a test segment once activated by the FSM/mem rising front, each V_i module acknowledges its end; this signal, V_end, reaches the Stimuli Generator CTRL unit.

The TSO and RTS files content is received at run-time from the tester control unit and read from memory, respectively; this operation is performed by means of 2 circular buffers, TSO and RTS buffers, and managed by two separated managements units. These units are in charge of continuously monitoring the buffer content in order to replace already used parts with new ones. This operation is done concurrently with the stimuli generation and application processes.

The TSO buffer is read by the Stimuli Generator CTRL unit and is word addressable; on the contrary, the RTS buffer has to be bit addressable since a variable number of bits can be requested for patterns reconstruction, depending on the associated horizontal occurrence. This information is provided by each H_i unit by means of a pattern mask and used by the Reconstr Unit whose output is the complete pattern value. When moving from a reduced test set to another, only the H_i and V_i are requested to be redesigned, accordingly with vertical and horizontal occurrence identified.

4. Experimental results

The developed experimental setup exploits a semi-automated tool chain that includes a Lexical Analysis and a Translation tools. The former is able to identify vertical and horizontal occurrences in a VCD file basing on user provided test segments, again provided in VCD; the latter is able to translate a recurrent test segment described in VCD language into a FSM. Both these tools are implemented in AINSI C (totally about 1,500 code lines). Fig. 7 shows the pattern analysis experimental setup.

As underlined in section 3, some of the tester components supposed to be embedded on FPGA do not require any redesign when changing the analyzed pattern (e.g., the buffer managers and the Stimuli Generator control unit. Therefore, fix parts are designed in VHDL and integrated with the FSM VHDL descriptions resulting from the pattern analysis phase. In our environment, we used a Xilinx Spartan3 XC3S1000 FPGA as reference for our evaluation and we used it for prototyping the Stimuli Generator design. This FPGA contains about 1M equivalent gates, including about 15,000 flip-flops and 500k SRAM bits; buffers are mapped over SRAM blocks available on FPGA. Table I shows the occupation of fix components mapped on the considered device.

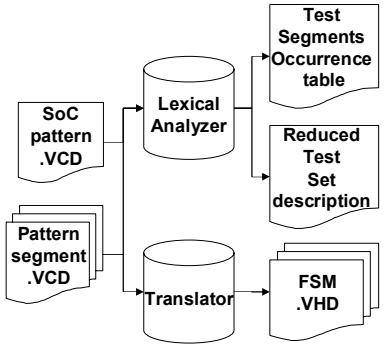

Fig. 7: Pattern analysis experimental setup.

Stimuli generator Component	Lut [#]	FF[#]
Stimuli Generator CTRL Unit	64	57
RTS Buffer Manager	41	38
TSO Buffer Manager	49	27
V_mux + H_mux + FSM/mem_mux	158	0
Reconstr Unit	345	0
Curr_stimuli_vector_reg	0	32
Total	**657**	**154**

Tab I: Components occupation on FPGA

It should be noticed that current FPGAs offer enough blocks to store the entire system including control unit, stimuli generator and monitoring unit modules and memories. Reducing the pattern size is a fundamental issue for such an implementation since it permits to maximize the number of vectors store in the FPGA memory that is currently represents a bottleneck for programmable devices.

The evaluation of the proposed methodology was done on a sample SoC including self-tested processor and memory cores. Low-cost test methodologies considered are:

a. Software-based Self-Test of microprocessors [4]
b. Programmable Built-in Self-Test of memories [11].

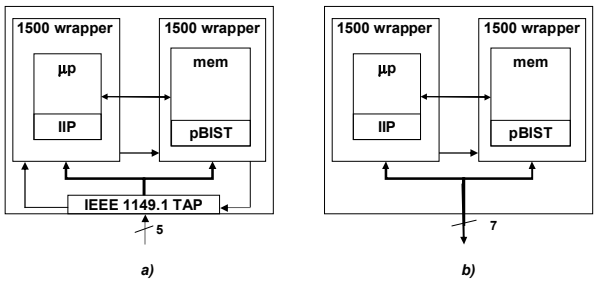

Fig. 8: the 2 investigated SoC test architectures.

Two versions of the SoC structure, shown in fig 8., have been implemented considering 2 test access mechanisms:

1. cores surrounded by an IEEE 1500 wrapper whose functionalities are controlled through a TAP controller compliant with the IEEE 1149.1 Standard
2. cores surrounded by an IEEE 1500 wrapper whose functionalities are directly controlled from the top-level of the SoC.

This methodology can be fruitfully extended to other protocols such as on-going works on IJTAG/SJTAG.

We assumed that the maximum supported tester frequency is 20 MHz while the test execution is supplied by an internal (or tester independent) 200 MHz source.

Considering processor Software-based Self-Test, results refer to the application of test procedures described in [4] for a Minimips RISC processor. Such a processor architecture fetches instructions and data codified on 32 bits; addresses are provided on a 32 bits bus, too. In the supposed environment, it means that at least 67 (32 instruction + 32 address + 3 control bits) clock cycles are requested for each program word to be serially transferred into a suitable embedded memory, therefore loading programs is the major cost for this test procedure application. The considered procedure accounts for 1,565 words and 7,162 clock cycles are requested for its execution once loaded. In [4], few details are given concerning results retrieval; we assume here that 256 data memory locations are used to store results, each one on 32 bits. Table II summarizes the results for this case study. The gain is up to about the 64% in case of pure IEEE 1500 access type.

SoC Access Structure	Vertical occ. [#]	Horizontal occ. [#]	Full pattern set [kbits]	Reduced pattern set [kbits]
IEEE 1500 + TAP ctrl	4 (5 ck)	2 (2 bits)	~527	~287
IEEE 1500	0	2 (5 bits)	~871	~313

Tab. II: test data volume reduction results for SBST test procedures application.

Considering memory testing, the results refer to a test procedure application exploits a Programmable BIST whose architecture is detailed in [11]; the management of this self-test architecture includes initialization, test activation and results retrieval phases. The memory test algorithm is stored in a microcode memory whose parallelism is 4 bits and address bus is 6 bits; therefore, the loading operation for a selected algorithm consists in serially shifting 64 sequences, each one composed of 12 bits. In this specific case, a 16M memory core with a 32 bits data parallelism is tested by applying a 12n march algorithm. Test execution results in about 5M clock cycles at tester frequency. The test data volume reduction is very high, mainly because of horizontal occurrence identified during autonomous memory test execution. Commodities like this one are commonly employed in the industrial practice to save tester memory when waiting for test completion; anyway, fairly speaking, an additional gain of about 30% is obtained concerning test procedure initialization either in case a TAP controller or solely IEEE 1500 wrapper are used. Initialization and execution phase gains are separately reported in table III.

SoC Access Structure	Vertical occ. [#]	Horizontal occ. [#]	Full pattern set [kbits]	Reduced pattern set [kbits]
IEEE 1500 + TAP ctrl	4 (5 ck)	2 (2 bits)	(INIT) ~4 (EXEC) ~20,132	(INIT) ~3 (EXEC) ~19
IEEE 1500	0	2 (5 bits)	(INIT) ~6 (EXEC) ~35,232	(INIT) ~4 (EXEC) ~19

Tab. III: test data volume reduction results for programmable memory BIST test procedures application.

Concerning the described two SoC test access mechanisms, the stimuli generator occupation ranges between the 17% and the 11% of the available FPGA resources, respectively. The difference is due to the different occurrence types identified, thus to the different FSMs included on FPGA.

5. Conclusions

This paper provides a methodology suitable to reduce the test data volume when low-cost test strategies are used. The illustrated method consists in the analysis of the test pattern set for identifying recurrent test segments. These segments are pruned from the pattern set and supposed to be reproducible in a hardware matter on the tester; therefore, the described strategy relies on the assumption that a FPGA is included in the used tester architecture to store a suitable stimuli generator.

Experimental results shows the effectiveness and the feasibility of the methodology for low-cost self-test strategies of microprocessor and memory cores.

6. References

[1] P.T. Gonciari et al. "Integrated test data decompression and core wrapper design for low-cost system-on-a-chip testing" IEEE International Test Conference, 2002, page(s):64 – 73

[2] Beck, M. et al. "Measures to improve delay fault testing on low-cost testers - a case study", IEEE VLSI Test Symposium, 2005, page(s):223 – 228

[3] Zorian, Y.; "Guest editor's introduction: what is infrastructure IP?", IEEE Design & Test of Computers, Volume 19, Issue 3, May-June 2002 Page(s):3 – 5

[4] M. Psarakis et al. "Systematic software-based self-test for pipelined processors", ACM/IEEE Design Automation Conference,2006, page(s):393–398

[5] A. Benso et al. "HD2BIST: a hierarchical framework for BIST scheduling, data patterns delivering and diagnosis in SoCs", IEEE International Test Conference, 2000, page(s): 892 – 901

[6] Stroud, C. E.; "A designer´s Guide to Built_In Self_Test", Kluwer Academic Publisher, 2002

[7] http://www.vsi.org/, 2006

[8] E.J. Marinissen et al. "Towards a standard for embedded core test: an example", IEEE International Test Conference, 1999, page(s): 696 - 705

[9] K. Chakrabarty, "Low-cost modular testing and test resource partitioning for SOCs", IEE Proceedings - Computers and Digital Techniques, Volume 152, Issue 3, 6 May 2005 Page(s):427 – 441

[10] IEEE 1149.1-1990 Standard Test Access Port and Boundary-Scan architecture

[11] D. Appello et al. "Exploiting programmable BIST for the diagnosis of embedded memory cores", IEEE International Test Conference, 2003, pp. 379-385

[12] E.J. Marinissen et al. "The role of test protocols in testing embedded-core-based system ICs", IEEE European Test Workshop 1999.