STESI: Testing wrapped IP cores using a dedicated Test Processor

Matthieu Tuna, Mounir Benabdenbi, Alain Greiner
Laboratoire LIP6, Dept ASIM
12, Rue Cuvier, 75252 Paris Cedex 05, France
Tel. (+33)1 44 27 65 28 - Fax. (+33)1 44 27 72 80
{matthieu.tuna, mounir.benabdenbi, alain.greiner}@lip6.fr

Abstract

This paper presents STESI, a software-based approach for testing SoCs containing wrapped IP cores. In the proposed approach, the test program is no more executed by the traditional ATE but by the SoC itself. The novel feature of the STESI approach is the use of a dedicated test coprocessor embedded on the SoC to test the remaining components. Using the ITC02 SoC benchmarks a comparison is done between the STESI architecture and a classical bus-based strategy.

Keywords

System on Chip Testability, Test Access Mechanism, Software-based Test, IEEE 1500, ATE

1 Introduction

With the advent of Systems on a Chip (SoCs), design methodologies are mainly driven by the Time-To-Market, and therefore are more and more based on the use of predesigned Intellectual Property (IP) cores. Design reuse is mandatory in these design methodologies, but test reuse is not easy to implement [1]. In fact, the testing of complex SoCs poses many challenges, like the need for an uniform test architecture for cores with different functionalities provided by different IP vendors.

To answer these challenges, several working groups have been spawned [2], [3], to define new standards and to simplify test integration. The IEEE 1500 (previously IEEE P1500) working group defines the way a core must be shipped to be tested when integrated in a SoC. The IEEE 1500 group specified a Wrapper [4] to be added around each IP core, and a Core Test Language (CTL) which enables the descriptions of the core test features. Note that the working group does not intend to standardize the way the core is accessed for test purpose. The SoC integrator is free to use the appropriate Test Access Mechanism (TAM). Most of the TAMs published so far [5] [6] [7][8] are based on the use of a dedicated test bus. The performance of this kind of test architecture is closely tied to ATE capabilities. As the gap between Automated Test Equipment (ATE) capabilities and chip complexity is getting larger, testing new SoCs requires expensive ATEs with high frequency, great accuracy and large memory. To deal with these limitations, a natural way consists in transferring some of the ATE capabilities into the SoC.

In this paper we present the STESI approach, a new test strategy applying this concept through the reuse of SoC existing resources. STESI stands for Software-based Test Environment for SoCs containing wrapped IP cores. With this approach, the SoC is now able to bring back test patterns from an external memory, apply these test stimuli to the different IP core I/Os, capture responses, and compare on-chip these results with the expected values. The test program is executed on-chip by the SoC. This test program includes test data (stimuli and expected responses) and test instructions defining what to do with the test data. Many benefits appear using this methodology. First, the test of an SoC does not depend on the ATE frequency, the test is applied at SoC speed. Second, an ATE great accuracy is no more needed since the comparison between captured test responses and expected ones are performed on-chip. Third, this approach offers great flexibility in developing the test program, and moreover it induces a minimum area overhead since SoC functional resources are reused.

In the following, we first briefly describe the basics of software-based SoC testing, and analyze the prior work. Next section presents the STESI strategy: the targeted SoCs and the test execution process. Then, some software features of the proposed approach are underlined followed by a study of the test co-processor internal architecture. Before concluding, first results on ITC'02 SoC benchmarks [9] are discussed and compared with a classical bus-based approach.

2 Prior work

The concept of reusing the embedded microprocessor for SoC test purpose is not new. In [10] A. Krstic et al. give a survey of the embedded software-based self testing paradigm. According to [10], the use of the embedded microprocessor to provide and analyze test data to/from IP cores relies on two main assumptions.

First, the microprocessor has been previously tested. Many approaches have been proposed to address this issue. Testing the embedded programmable core can be done using hardware based approaches, [11], [12] and/or software based ones [13], [14].

Second, the system bus and global interconnects have been also previously tested. Using the embedded microprocessor, testing the system level interconnects can be addressed using techniques as described in [15], [16], [17].

Relying on these assumptions, significant software based SoC test architectures have previously been published.

In [18], the proposed methodology uses the embedded microprocessor to apply test patterns to the main IP cores and analyze responses through the system bus. The test program is run by the microprocessor and each IP has a wrapper associated that includes test control and buffers. This approach requires the embedded cores to be full scan and an access to their netlists. This is hardly the case when using proprietary IPs.

In [19] the described RASBus architecture takes advantage of the system bus and the embedded microprocessor. In this architecture, the access to the core inputs/outputs is done through a proprietary Test Access Interface.

In [20], a similar approach is presented. The main concept is based on the use of the embedded microprocessor/memory pair to test the other SoC components. Test data is downloaded from the external ATE, using DMA techniques, into the system memory. The microprocessor uses these data to test the main cores. To reach this objective, the microprocessor has to control the main components in order to provide test access path to the targeted core. Test responses are transferred to a MISR for evaluation.

The approaches described in [18] and in [20] are based on the use of the embedded microprocessor. However their flexibility is limited since the test engineer must have a full knowledge of the IP cores internal architecture.

In [21], a processor for Embedded Test is presented. This processor controls the at speed test of a complex system including cores with different test strategies. However, this interesting approach does not fully use the internal resources of the SoC and is limited by the internal memory size that contains the test program.

3 STESI methodology

In this section we first see some hardware requirements, the SoC template targeted by STESI to perform on-chip the application of test patterns to wrapped IP cores. Then, we will see how the test is performed, for go/no-go testing and for failure analysis.

3.1 STESI targeted SoCs

STESI methodology can be applied to test SoCs having the following characteristics (see figure 1):

- The SoC must have an interconnect supporting initiator/target scheme.

- The SoC is shipped with an external RAM controller with a 32-bit interface. During functional SoC operation, the external controller is used to plug extra-memory or peripheral. During SoC testing, the pins of this interface are connected to an extra-memory containing as many test programs as wrapped IP cores.

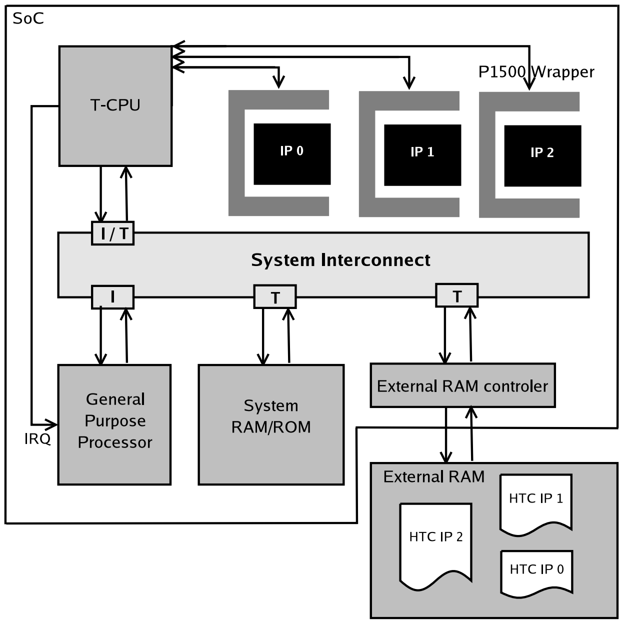

Figure 1. STESI Architecture overview

- IP cores to be tested are wrapped. STESI approach can drive many wrapper types (IEEE 1500, boundary-scan and even some bist controller engines). However, the following of the paper focuses on the test of IEEE 1500 wrapped cores. For each bloc (local or third-party), test informations and test patterns are supposed available.

- The SoC is equipped with an embedded 32-bits microprocessor used for general purpose (GPP).

Besides this SoC scheme, STESI introduces a single new hardware component: a test coprocessor dedicated to SoC testing called TCPU. It has two interfaces. On one side, it is a memory-mapped peripheral for the interconnect and can thus be addressed like any resource by the embedded microprocessor. On the other side, it is a IEEE 1500 pattern delivery TAM for wrapped cores. As an initiator the TCPU can directly address the external RAM and thus, read test programs stored in. As a target, the TCPU receives commands emitted by the GPP. TCPU can interrupt the GPP setting up an IRQ.

3.2 Test execution

In STESI, the test is performed by the GPP/TCPU pair. The TCPU is in charge to process test programs. Each tested IP have his own test program, available in a format called HTC, specific to the TCPU. Each HTC file is stored in the external memory (see figure 1). A HTC test program is a sequence of test instructions fetched and executed by the TCPU. Those test instructions consist on (among others) applying test patterns, capturing test responses, and comparing with an expected value. TCPU enables concurrent testing of many cores in order to minimize the test application time. During the test process, for each tested IP core, the TCPU stores informations such as the status (test pass, test fail, test in progress), program counters, etc. These informations can be accessed at any time by the GPP. When a failing comparison is detected or when the core test is done, an IRQ to the GPP is set up.

The GPP acts as a chief orchestra, controlling test programs execution. It launches the test on the targeted IP by sending to the TCPU a start request containing the corresponding IP number. The TCPU can then start the test of this IP as described above. During the test process, the GPP can at any time send any request to the TCPU to consult test informations about an IP. The GPP can monitor test execution through polling. To avoid overload on the system interconnect the GPP can also wait for an IRQ to be emitted by the TCPU. When the core test is over, the GPP collects informations about the test and stores them out of the SoC (writing in a special address in the external memory for example). STESI allows test for production go/no-go testing as well as advanced failure diagnosis. The difference between these two mode (go/no-go and diagnosis) is based on the volume of information stored out of the SoC by the GPP. While a minimal information like OK/KO for go/no-go testing is enough, more informations are needed by the test engineer to target a fault. The GPP program, called the **"Master Test Program"**, can be stored in the embedded system memory as well as in the external memory.

To summarize, in the STESI approach, the GPP has a central role, since no ATE drives the test execution. It has a complete control on the test execution and is the interface between the test program execution and the SoC external test environment.

4 STESI Software Toolbox

This section depicts the STESI software test flow, from standard CAD file generation (STIL [22], CTL [2] [4]) to advanced failure diagnosis. We linger on tools and file formats created in the STESI project, enabling interactions with usual CAD test tools like ATPG or fault simulator. The figure 2 shows this flow from test programs generation to failure analysis.

For production go/no-go testing, the only required inputs of the flow are the STIL or CTL files of each core to be tested. However, to enable the diagnosis process the netlist of the faulty core must be provided.

4.1 STIL to HTC

Nowadays most of IP cores test informations and test patterns are delivered in STIL or CTL formats. Therefore, it is required to convert this format in a TCPU executable test program called HTC. This conversion is automated by GenSTELA (for STEsi LAnguage Generator). The generated HTC file includes a suite of 32 bits instructions. The execution is completely sequential, without any loop or jump. An instruction is divided in 2 fields: 8 bits of opcode and 24 bits of data. The opcode specifies what to do with the data. Table 4.1 shows several instructions: the opcode, the kind of the value in the data field and an explanation about the instruction. For each wrapper connected to the TCPU, a couple of 24 bits registers is associated. The figure 5 shows the connection

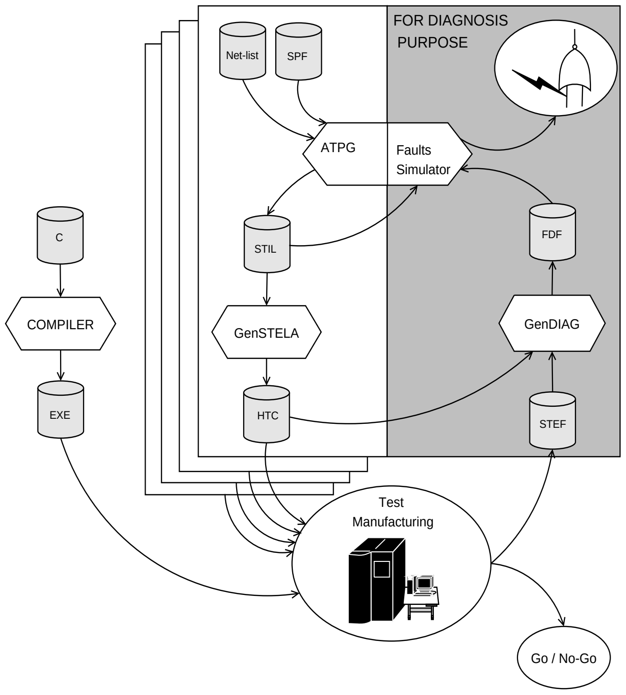

Figure 2. STESI Soft Flow

between the wrapper and these two registers. The SHIFTINOUT instruction uses these two registers. The first one is used to shift in, in a serial way, a test pattern (SHIFTINOUT instruction data field), while the second is used to shift out the produced test response. The COMPARE instruction compare the 24 bits contained in the data field with the 24 bits stored in the second register.

Obviously, STESI is not limited on sending test vectors whose size is less or equal to 24 bits. If a vector is longer than 24 bits (that is generally the case) this vector is cut into 24 bits chunks. If the test vector size is not a multiple of 24, the last chunk is completed with padding bits. Thus, applying one test vector requires a sequence of SHIFTINOUT instructions. The figure 3 shows the conversion of a part of a STIL file to the corresponding (human readable) HTC pro-

OPCODE	DATA	comments
LOADWIR	wir value	load the Wrapper Instruction Register of the target core
UPDATE	none	sets up the Update signal
CAPTURE	none	sets up the Capture signal
SHIFTINOUT	test vector	this instruction is designed to load 24 bits of a test vector and unload 24 bits of produced response
COMPARE	expected value	compare the expected value with the 24 bits get back by a shiftinout instruction

Table 1. Several HTC instructions

gram. This example is based on the translation of the use of the classical "load_unload" function which role is to load the vector "$i_0 i_1 i_2 i_3 .... i_{49}$" through the **WSI** port and to compare the produced vector with "$o_0 o_1 o_2 o_3 .... o_{49}$" unloaded through the **WSO** port.

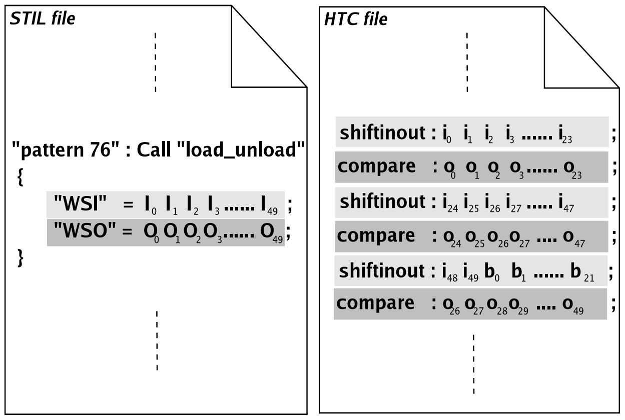
STIL file
"pattern 76" : Call "load_unload"
{
"WSI" = I0 I1 I2 I3 ...... I49 ;
"WSO" = O0 O1 O2 O3...... O49;
}
HTC file
shiftinout : i0 i1 i2 i3 ...... i23 ;
compare : o0 o1 o2 o3...... o23 ;
shiftinout : i24 i25 i26 i27 ..... i47 ;
compare : o24 o25 o26 o27 .... o47 ;
shiftinout : i48 i49 b0 b1 ...... b21 ;
compare : o26 o27 o28 o29 .... o49 ;

Figure 3. STIL to HTC format conversion

4.2 Master Test Program

The master test program executed by the GPP supervises the execution of the HTC programs. This master test program is written in assembler or high-level languages like C. It is (cross-)compiled and the resulting binary is loaded into an internal or external memory. Easiness of writing in high-level languages allows designing master test program as complex as desired: smart test plan scheduling, restarting test program on special part, etc. However, this test program should include the following four steps:

1. Starting test process by sending to the TCPU the HTC programs addresses. Selected IPs are tested concurrently.
2. Wait TCPU interruption(s).
3. Collect more or less informations according to selected mode, i.e. Go/No-Go mode or Diagnosis mode.
4. Store informations out of the SoC through the external RAM controller.

A simple Master Test Program can be designed as presented below:

```
/*
 *        MASTER TEST PROGRAM
 *              PHASE (1)
 *
 * launch test on different IPs,
 * AD_PROG_HTCx : HTC program start address,
 *                stored in external memory
 */
launch_test(AD_TCPU, AD_PROG_HTC0);
launch_test(AD_TCPU, AD_PROG_HTC1);
launch_test(AD_TCPU, AD_PROG_HTC2);

 *              PHASE (2)
 *
while(wait_TCPU_Int);

 *              PHASE (3)
 *
 * information collect
 */
IP_NUM  = get_ip_stopped();
info1   = get_status(IP_NUM);
info2   = get_res(IP_NUM);
info3   = get_epc(IP_NUM);

 *              PHASE (4)
 *
 * send informations to outside
 */
*(AD_OUTSIDE + 1) = info1;
*(AD_OUTSIDE + 2) = info2;
*(AD_OUTSIDE + 3) = info3;
```

4.3 Diagnosing test failures

This section focuses on failure analysis, and so, concerns only the core whose netlists are available. In traditional testing when ATE drives test execution, when a device fails testing, the ATE can store the failure informations into a file. Let us call this file the failure data file (FDF). Thanks to three files (the netlist, the test patterns, the failure data file) a fault simulator tool with diagnosis capability can determine the cause of the failing patterns and generates a diagnosis report. As in STESI no ATE drives the test execution, the generation of this FDF is done using GenDIAG. The figure 2 shows the flow for diagnosing manufacturing test failures. During test execution the GPP stores out of the SoC the test failure informations in a file, in a proprietary format (STEF). Using the HTC file of the faulty IP core and the corresponding STEF file, GenDIAG can generate the required failure data file (FDF).

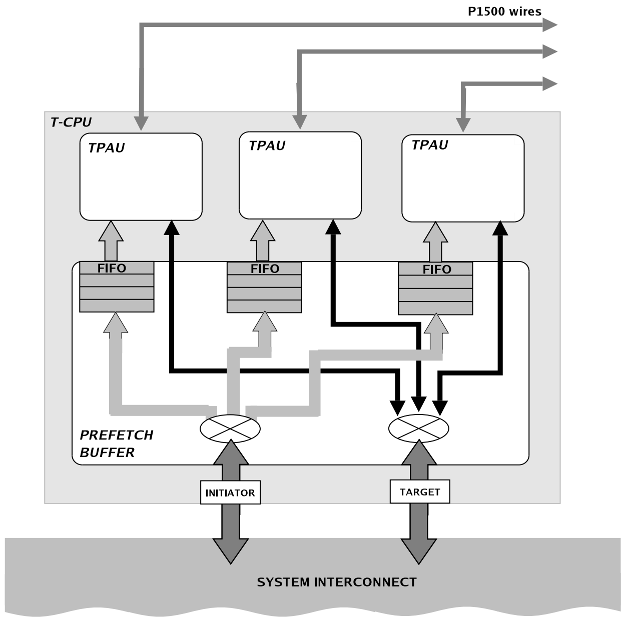

Figure 4. TCPU Internal Architecture

5 TCPU internal architecture

This section focuses on the TCPU internal architecture in more details (see figure 4). This test coprocessor includes two major kinds of components namely a **Prefetch-Buffer** (PB) and a **TPAU** (standing for Test Protocol Associated Unit). The number of TPAU is equal to the number of IPs to be tested. Each TPAU is dedicated to one wrapped IP core. All the TPAUs shares the Prefetch Buffer interface. The communication between PB and TPAUs is enabled by a FIFO protocol. The role of the PB is to get the test programs from the external RAM, and feed the right TPAU with the right instructions.

5.1 The TPAU

The TPAU is the unit that execute the HTC program and convert it to a IEEE 1500 data stream. It has two interfaces: on one side, two communication ports, one to get the test program and one for configuration purpose. On the other side, the TPAU is connected to the wrapper, through the IEEE 1500 Serial Interface Layer (SIL) (see figure 5). The TPAU is an assembly of two main components:

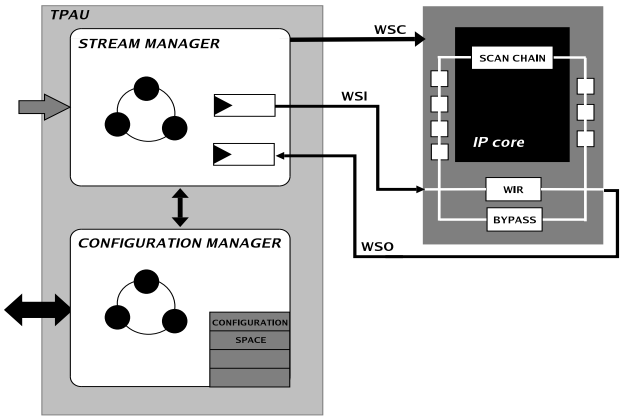

Figure 5. TPAU Internal Architecture

- the Stream Manager: the unit that executes the HTC code. This unit is connected to the wrapper. It convert test program in IEEE 1500 test data stream.
- the Configuration Manager: when the GPP requests some informations about the test of a wrapped core, it sends a read request to the configuration manager of the corresponding core. All the test informations are stored in the configuration space.

5.2 The Prefetch Buffer

The role defined for the PB is to provide an interface between the system interconnect and the TPAUs. As described previously two type of connections are required: one for the test program, one for configuration purpose. Thus, two connection ports are available:

- Initiator port: through this port the PB brings back the test program from the SoC external RAM to feed the right FIFO. The PB ”pre”-fetch chunks of test program in order to minimize TPAU idle time. A round Robin algorithm selects the FIFO to be fed (see figure 4). Through this port, it acts as a DMA.
- Target port: this port is memory-mapped in the system, and so, provides a GPP access to the desired TPAU (configuration manager). Each TPAU has a specific address in the memory mapped environment. The PB receives all the requests, and dispatch these requests to the right TPAU.

The PB is generic, depending on:

- The number of cores to be tested.
- FIFOs depth. This parameter is significant since it allows making trade-offs between area overhead and test application time.
- The system interconnect protocol.

6 Experimental results

First results have been obtained with a SystemC-based simulation platform containing both hardware and software components. Simulations made are bit-accurate and cycle-accurate. The GPP model used is a MIPS R3000 one. The connections between the TCPU and the wrapped IPs are established thanks to dedicated test wires. The STESI approach have been applied to IP cores wrapped with the IEEE 1500 Serial Interface Layer. Table 6 shows the test application time, using the STESI architecture, applied to four SoC ITC02 benchmarks. The results are compared to those of a bus-based TAM strategy: TR-Architect [23] with a 32 bits TAM. To perform the SoC test, the proposed approach requires from 3 to 6 times more test cycles than in a traditional bus-based strategy. It is an expected result that the STESI methodology induces far more test cycles than TR-Architect. This is mainly due to the handshake protocol between the active components through the system interconnect. However, as these results are provided as number of cycles, one must take into account the test application frequency. In the STESI method the frequency is the functional SoC one, while in the TR-Architect approach the frequency used is the tester’s one. Moreover the first results obtained with the IEEE 1500 Parallel Interface Layer shows that the STESI approach is twice consuming than TR-Architect in term of cycles. Thus, the global test time, in seconds, can then be in favor of the STESI approach. As the test comparison is made on-chip, no yield loss is due to the tester lack of accuracy, what can appear in a traditional bus-based methodology.

Reusing the functional resources for test purpose minimize the need of extra hardware dedicated for test. No dedicated test pins are added, as the external RAM interface is reused. This implies that in STESI, the TAM bandwidth is not scalable. However, this choice makes STESI ATE independent.

Traditional bus-based strategies impose to make one chip test program, the tests of the main cores being merged in a single complex program, hard to develop and not easy to modify. STESI offers much more flexibility since each test program execution of each core is independent from the other.

SOC	**STESI**	**TR-Architect**	$\frac{STESI}{TR-A}$
d695	211,874	44,307	4.78
p22810	2,333,326	458,068	5.09
p34392	6,902,196	1,010,821	6.08
p93791	6,379,717	1,791,638	3.56

Table 2. Test application time (cycles)

7 Conclusion

This paper presented STESI,a new software-based methodology designed for testing SoCs including wrapped IP cores. Using the SoC internal resources, the test can be done thanks to a dedicated test coprocessor. No dedicated scan pins are needed. The test program is executed by the on chip microprocessor at SoC speed. Compared to a traditional bus-based strategy, this new approach allows the test process not to be bound to the tester frequency or accuracy. This enables the use of lower cost ATEs.

The proposed method offers also optimum flexibility for the test program development since the test program can easily be adapted to fit the best test mode (production, diagnosis...). It can even be developed after the chip has been sent to foundry and can easily be ported to a different SoC containing a different microprocessor.

References

[1] Yervant Zorian, Erik Jan Marinissen, and Sujit Dey. Testing Embedded-Core Based System Chips. In *Proceedings IEEE International Test Conference (ITC)*, pages 130–143, Washington, DC, October 1998.

[2] IEEE P1500 Web Site. http://grouper.ieee.org/groups/1500/.

[3] VSI Alliance Web Site. http://www.vsi.org/.

[4] Erik Jan Marinissen et al. On IEEE P1500's Standard for Embedded Core Test. *Journal of Electronic Testing: Theory and Applications*, 18(4/5):365–383, August 2002.

[5] Erik Jan Marinissen et al. A Structured And Scalable Mechanism for Test Access to Embedded Reusable Cores. In *Proceedings IEEE International Test Conference (ITC)*, pages 284–293, Washington, DC, October 1998.

[6] Prab Varma and Sandeep Bhatia. A Structured Test Re-Use Methodology for Core-Based System Chips. In *Proceedings IEEE International Test Conference (ITC)*, pages 294–302, Washington, DC, October 1998.

[7] Lee Whetsel. Addressable Test Ports: An Approach to Testing Embedded Cores. In *Proceedings IEEE International Test Conference (ITC)*, pages 1055–1064, Atlantic City, NJ, September 1999.

[8] Mounir Benabdenbi, Walid Maroufi , and Meryem Marzouki. CAS-BUS: A Test Access Mechanism and a Toolbox Environment for Core-Based System Chip Testing. *Journal of Electronic Testing: Theory and Applications*, 18(4/5):455–473, August 2002.

[9] Erik Jan Marinissen, Vikram Iyengar, and Krishnendu Chakrabarty. ITC'02 SOC Test Benchmarks Web Site. http://www.extra.research.philips.com/itc02socbenchm/.

[10] A. Krstic, L. Chen, W.-C Lai, K.-T Cheng, and S. Dey. Embedded Software-Based Self-Test for Programmable Core-Based Designs. *IEEE Design and Test of Computers*, pages 18–26, july-august 2002.

[11] J. Shen and J. A. Abraham. Native Mode Functional Test Generation For Processors with Applications to Sel Test and Design Validation. In *Proceedings IEEE International Test Conference (ITC)*, Washington, DC, October 1998.

[12] Ken Batcher and Christos Papachristou. Instruction Randomization Self Test for Processor Cores. In *Proceedings IEEE VLSI Test Symposium (VTS)*, pages 34–40, Dana Point, CA, April 1999.

[13] N. Kranitis, G. Xenoulis, D. Gizopoulos, A. Paschalis, and Y. Zorian. Low-Cost Software-Based Self-Testing of RISC Processor Cores. In *Proceedings Design, Automation, and Test in Europe (DATE)*, Munich, Germany, March 2003.

[14] F. Corno, M. Sonza Reorda, G. Squillero, and M. Violante. On the Test of Microprocessor IP Cores. In *Proceedings Design, Automation, and Test in Europe (DATE)*, Munich, Germany, March 2001.

[15] Xiaoliang Bai, Sujit Dey, and Janusz Rajski. Self-test methodology for at-speed test of crosstalk in chip interconnects. In *Proceedings of the 37th conference on Design automation*, pages 619–624. ACM Press, 2000.

[16] W.-C. Lai, J.-R. Huang, and K.-T. Cheng. Embedded-Software-Based Approach to Testing Crosstalk-Induced Faults at On Chip Buses. In *Proceedings IEEE VLSI Test Symposium (VTS)*, pages 204–209, Marina del Rey, CA, May 2001.

[17] E.Cota E.Correa, R.Cardozo. Testing the wrappers of a network on chip: a case study. In *LATW'2003 - IEEE Latin-American Test Workshop*, February 2003.

[18] Jing-Reng Huang, Madhu K. Iyer, and Kwang-Ting Cheng. A Self-Test Methodology for IP Cores in Bus-Based Programmable SOCs. In *Proceedings IEEE VLSI Test Symposium (VTS)*, pages 198–203, Marina del Rey, CA, May 2001.

[19] Sungbae Hwang and Jacob A. Abraham. Microprocessor Based Test Structure for SOC. In *Digest of Papers of IEEE International Workshop on Testing Embedded Core-Based Systems (TECS)*, pages 4.3–1–7, Marina del Rey, CA, May 2001.

[20] C. A. Papachristou, F. Martin, and M. Nourani. Microprocessor based testing for core-based system on chip. In *Proceedings of the 36th ACM/IEEE conference on Design automation conference*, pages 586–591. ACM Press, 1999.

[21] L. Carro E. Cota, F. Brisolara. Met: An embedded processor for test controlling. In *IEEE International Workshop on Testing Embedded Core-based System-Chips*, 2001.

[22] IEEE 1450 Web Site. http://grouper.ieee.org/groups/1450/.

[23] Sandeep Kumar Goel and Erik Jan Marinissen. Effective and Effi cient Test Architecture Design for SOCs. In *Proceedings IEEE International Test Conference (ITC)*, pages 529–538, Baltimore, MD, October 2002.