STESOC: A Software-Based Test-Access-Mechanism Controller

Matthieu Tuna, Mounir Benabdenbi, Alain Greiner
Laboratoire LIP6, Dept ASIM
University of Pierre et Marie Curie, Paris VI
{matthieu.tuna, mounir.benabdenbi}@lip6.fr

Abstract

Software-based test of SoCs consists in testing IP cores using embedded processor cores. Previously proposed solutions are usually ad-hoc. Therefore, this paper presents STESOC, a software-based Test Access Mechanism for SoCs containing standard-wrapped IP cores. Under the control of the embedded microprocessor, a dedicated test-coprocessor tests the remaining components. Using the ITC02 SoC benchmarks a comparison is done between the STESOC 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 pre-designed Intellectual Property (IP) cores. IP cores such as microprocessors, memories, DSP, peripherals, or dedicated blocks are bind together to make complex SoCs. This deep integration has consequences for the test process. Poor test accessibility for external testers increase test application time, and huge memory is required in order to store test data volume. Moreover at-speed testing is becoming a very challenging task since the gap between the tester's frequencies and SoC frequencies is getting larger.

A solution to alleviate the role of the Automated Test Equipment (ATE) can be found in hardware Built-In Self-Test (BIST) techniques. Internal hardware test generators and test response analyzers are used to generate and apply test patterns at the speed of the circuit. Unfortunately, each BISTed module inserted in the system may result in performance degradation in terms of total area, operating frequency and power consumption.

With the SoC paradigm, alternative solutions like Software-Based Test methodology is emerging. Software-Based Test (SBT) consists in the execution of a program by an embedded microprocessor to test embedded cores. Usually, the test program is initially downloaded into the internal memory thanks to a low-cost external equipment. First, the embedded microprocessor tests itself by executing a self-test program, then, it can be used as a pattern generator and response analyzer to test the others cores of the system.

As for the test of microprocessors, system interconnects and memories have been well studied and convenient SBT solutions are provide. However, there is a lack of proposed solutions to test other kind of cores. Those remaining cores are application specific, consequently, proposed software-based test strategies are often ad-hoc.

To answer the needs for a uniform test architecture for cores provided by different IP vendors, several working groups have been spawned [1], [2]. They define new standards in order to simplify test integration. The IEEE 1500 working group defines the way a core must be shipped to be tested when integrated in a SoC. This group specified a Wrapper [3] 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).

In this paper we investigate the use of a Software-based TAM relying on the IEEE 1500 standard. This TAM is embodied as a dedicated co-processor drive by the embedded processor. On one side, it can be addressed by the embedded microprocessor, and, on the other side it is a IEEE 1500 pattern delivery TAM for wrapped cores.

In the following, we first briefly describe the basics of software-based SoC testing, and analyze the prior work. Next section presents the STESOC strategy: the targeted SoCs and the test execution process. Then, a study of the test co-processor internal architecture is presented followed by some software features of the proposed approach. First

results on ITC'02 SoC benchmarks [4] are discussed and compared with a classical bus-based approach. Before concluding, some future works enhancing this approach will be presented.

2 Software-Based Test

The concept of reusing the embedded microprocessor for SoC test purpose is not new. In [5] A. Krstic et al. give a survey of the embedded software-based testing paradigm. Software-based test assumes the SoC have at least one embedded processor. This is generally the case since, nowadays, almost all SoC contains an embedded processor core surrounded by memories and others cores. In Software-based Test, the general purpose processor (GPP) has a central role in the test process since it acts as a chief orchestra. Therefore, no specific test controller is required. First, the embedded microprocessor tests itself by executing a self-test program, then, it is used to test the system interconnect, embedded memories and others remaining cores of the system.

Integrated IP cores can be divided into two sets. The first set is the general purpose cores (GP cores) such as microprocessor, memories or interconnect. There are almost always embedded in the SoC. The other set contains IP cores that are application specific (ASIC cores), such as JPEG decoder, Media Access Controller (Ethernet), USB controller core etc. Their use strongly depends on the need of the targeted application. Thus, four different kinds of software-based tests can be outlined: (i) self-test of processor core, (ii) test of the system interconnect, (iii) test of embedded memories and (iv) test of remaining cores (ASIC cores).

Processor Test. Different methodologies have been proposed for the development of the self-test program of processor core. Two different strategies can be distinguished. In one hand, the self-test program is generated automatically like in [6] and [7]. In [6] the self-test program is automatically generated off-chip whereas in [7] the authors propose an on-chip generation of the self-test program. In the other hand, in [8] the authors achieve higher fault coverage, but the self-test program is designed by the test engineer. The self-test program is a set of small specific self-test routines, based on compact loops of instructions, developed for each processor component.

Interconnect Test. Using the embedded microprocessor, testing the system level interconnects can be addressed using techniques as described in [9], [10]. In [10] the authors propose a Functionally Maximal Aggressor (FMA) test in order to test cross-talk effects that can occur in the bus. The test patterns are generated under system constraints (address system mapping) in order to avoid over-testing that lead to yield loss. The processor execute the test program, i.e. write on the bus to a destination core, the destination core capture the response, and the processor read this response and determine if the bus is faulty or not.

Memory Test. Most of memories embedded in a SoC are word-oriented memory (WOM). WOM tests have been presented in [11]. Those tests are very straightforward to be implemented using the processor. In [12], Tehranipour et al. propose a program written in assembly language implementing the 9N test algorithm. Besides, Rajsuman propose in [13] the use of March algorithm.

ASIC core Test. Microprocessors, interconnects, memories etc. have well-known structures. So, efficient software-based tests are provided. However, finding a uniform strategy to test all the remaining specific cores, while a high fault coverage is achieved is a hard task. Papachristou tackled the problem in [14]. He proposed a method implementing a Bypass approach. The key idea of the Bypass approach is to use existing circuitry between cores to vehicle test patterns to the target one. Ad-hoc circuitry is added for each core to enable the Bypass mode, and a sophisticated strategy is elaborated to create core input paths for pattern application and output paths for response capturing. Moreover, a synchronization of test packets must be done for cores connected to different input/output paths. In [15] the author assumes that, due to IP protection, only instruction set, limited architectural and test informations of the core are available. Thus, Tehranipour proposed a systematic approach to generate test programs. The embedded microprocessor is used to generate pseudo-random patterns. Because of the random nature of patterns, low fault coverage is achieved (86.1% for SPI core and 81.3% for HPI core). Furthermore, since the interconnect is used to vehicle the test patterns, a connection between the tested IP core and the system interconnect is required. In order to improve the fault coverage, in [16] Huang combines weighted random test patterns and a small set of deterministic test patterns. Moreover, each core is surrounded by a special test wrapper that supports pattern delivery. It contains buffers to store scan data and test support logic to control scan shifting. This technique implies a connection of each tested IP with the system bus. Moreover, the structural netlist of the core is supposed available, this is hardly the case when using proprietary IPs.

To recap, proposed techniques to test those cores are generally ad-hoc. These techniques require extra specific circuitry, knowledge of the IP cores internal architecture, connection to system bus and often, the use of pseudo-random patterns lead to low fault coverage.

3 STESOC methodology

The IEEE 1500 working group defines a wrapper [3] to be added around each IP core, and a Core Test Language (CTL) which enables the descriptions of the core test features. Use of such standard guarantees interoperability between core provider and core user, and might be popular especially for hard cores. Supporting such standards can alleviate the drawbacks of solutions describes before. No ad-hoc extra circuitry around each core need to be added since the test interface is standardized. If a dedicated TAM is provided, the core do not necessary require a connection to the system bus. And, since the test patterns are provided with the IP core, high fault coverage can be assumed.

In this section we first see some hardware requirements, the SoC template targeted by STESOC to perform on-chip the application of test patterns to wrapped IP cores. Then, we will see how the test is performed.

3.1 STESOC targeted SoCs

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

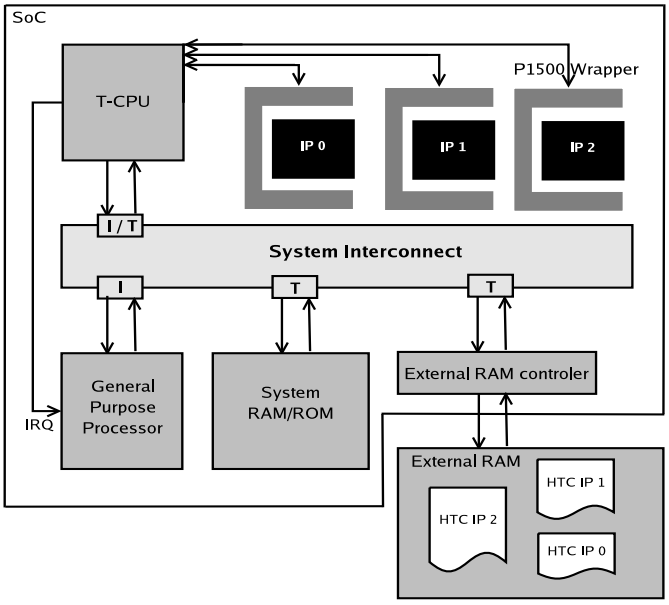

Figure 1. STESOC Architecture overview

- The SoC is equipped with an embedded microprocessor used for general purpose (GPP).
- 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.
- IP cores to be tested are wrapped. STESOC approach can drive many wrapper types (IEEE 1500, boundary-scan and even some BIST controller engines). However, the following of the paper focuses on test of IEEE 1500 wrapped cores. For each bloc (local or third-party), test informations and test patterns are supposed available.

Besides this SoC scheme, STESOC introduces a single new hardware component: a test coprocessor entirely 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 warn the GPP by the way of an Interrupt Request (IRQ).

3.2 Test execution

In STESOC, test is performed by the GPP/TCPU pair. The TCPU is in charge to process test programs. Each tested IP have its 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). HTC test programs are fetched and executed by the TCPU, it contains scan data (test patterns) as well as test control informations. Test control consist on (among others) control the scan shifting, applying test patterns and capturing test responses. Test responses are shifted in the TCPU and compacted by the way of an internal MISR. 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 done, test in progress), program counters, etc. These informations can be accessed at any time by the GPP. 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 is in charge to launch the test on the desired core by sending to the TCPU a start request containing the corresponding IP number. Thus, the TCPU can 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 a core. 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 the computed response and compare it with the expected signature of the core. Thus, one can know which IP core is fault free or not.

4 TCPU hardware architecture

This section focuses on the TCPU internal architecture in more details (see figure 2). 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 core is assigned to one TPAU. All the TPAUs shares the Prefetch Buffer interface. The role of the PB is to get the test programs in order to feed the right TPAU.

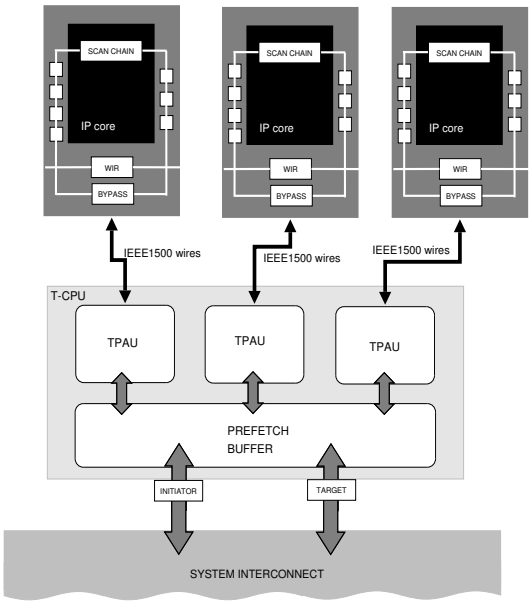

Figure 2. TCPU Internal Architecture

The TPAU is the unit that execute the HTC program and convert it to a IEEE 1500 data stream. The TPAU controls the scan shifting, the application of test patterns and the capture of test responses. The TPAU brings back response vectors that are compacted in an internal MISR. At the end of the test, one signature characterizing the core has been computed. The response compaction allows a drastic reduction of test data amount. The TPAU is associated to one IP core and so, one test protocol. For more simplicity only IEEE 1500 is evoked, but the TPAU can be Boundary-Scan compliant, or any specific test protocol, and can also drive some BIST engine.

The Prefetch Buffer provides an interface between the system interconnect and the TPAUs. As described previously two connection ports are available. Through the initiator port the PB brings back the test program from the SoC external RAM to feed the right TPAU. A round Robin algorithm select the TPAUs to be fed (see figure 2). Through this port, it acts as a DMA. The target port is memory-mapped in the system, and so, provides a GPP access to the desired TPAU. 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.

5 Test Programs

Two kind of test programs are executed. One by the embedded processor, several by the test co-processor unit (TCPU). The first one, called the *Master Test Program*, managing the test process of the IP cores. The seconds, called *HTC Test Programs*, executed by the TCPU, contains test data and test control of the corresponding IP. Each tested IP core has its own HTC test program.

HTC Test Programs. 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 STEsoc LAnguage Generator). At the moment, GenSTELA only parses a subset of STIL to generate the corresponding HTC file. This file includes a sequence of 32 bits instructions. The execution is completely sequential, without any loop or jump in order to reduce the hardware needs, and thus, TCPU size.

The Master Test Program. It is executed by the GPP supervising 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 memory. Easiness of writing in high-level languages allows designing a 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 three steps:

1. Starting test process by sending to the TCPU the HTC programs addresses. It starts the test execution of the selected IP cores. Selected cores are tested concurrently.

2. Wait TCPU interruption(s) indicating the end of the test process of cores.

3. Read the computed signature and compare it with the expected one.

6 Experimental results

The results are presented in terms of test application time and not in terms of fault coverage. Because in STESOC's strategy the fault coverage only depends on the test patterns delivered with the IP core. However, one can easily assume that in case of full-scanned IP core using provided deterministic test patterns a high fault coverage is achieved.

ITC'02 benchmarks [4] were used to show the test application time. The results are compared to those of a classical bus-based TAM strategy: TestRail optimized by the TR-Architect software toolbox[17]. The selected results for TR-Architect are those using a 32-bit channel width. Nevertheless, we can point out that in STESOC, no dedicated test pins are added on the SoC since the functional external-RAM interface is reused. The test pins overhead of STESOC is null, whereas, at least 65 dedicated test pins need to be added for TR-Architect (32 scin, 32 scout and test control pins).

First results have been obtained with a SystemC-based simulation platform containing both hardware and software components. Simulations done are bit-accurate and cycle-accurate. The GPP model used is a MIPS R3000 five-stage pipeline, written in SystemC as well as the TCPU model. An embedded RAM is added to store the Master Test Program, while an external RAM contains all the HTC test programs. Those latter components are connected via a VCI-compliant crossbar interconnect. For each module describes in the benchmark a corresponding SystemC module is added in the platform surrounded by an IEEE 1500 wrapper. The connections between the TCPU and the wrapped IPs are established thanks to IEEE 1500 dedicated test wires. Five ITC'02 benchmarks were used, test application time is presented in number of cycles. Each module of those benchmarks can be seen as a soft core or as a hard core. A hard core module implies that the number of the scan chains as well as their length cannot be changed. The module contains fixed-length scan-chains. Whereas for soft core modules, scan chains are not designed yet and can be optimized at SoC-level test architecture. Thus, the module contains flexible-length scan-chains. Table 1 shows the test application time (in terms of cycle number) for fixed-length scan-chains modules. The Table 2 presents experimental results for flexible-length scan-chains. To perform the SoC test, the proposed approach requires from 2 to 3 times more test cycles than a traditional bus-based strategy. In case of flexible-length scan-chains modules, STESOC requires only a factor less than 2 compared to TR-Architect. It is an expected result that the STESOC methodology induces far more test cycles than TR-Architect. This is mainly due to two issues. First, TR-Architect makes an architectural exploration to find the best test strategy in order to reduce the overall test application time. Whereas STESOC is a "plug&play" approach and for instance, the test plan schedule can be developed after the chip has been sent to foundry. The second issue of lost cycles is in the handshake protocol between the active components through the system interconnect. Nevertheless, as these results are provided as number of cycles, one must take into account the test application frequency. In the STESOC method the frequency is the functional SoC one, while in the TR-Architect approach the frequency used is the tester's one. Thus, the global test time, in seconds, can then be in favor of the STESOC approach. Moreover, 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 STESOC, the TAM bandwidth is not scalable. However, this choice make STESOC ATE independent.

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

SOC	STESOC	TR-Architect	$\frac{STESOC}{TR-A}$
d695	47,271	21,690	2.18
g1023	47,936	16,855	2.84
p22810	723,220	226,640	3.19
p34392	1,109,078	552,746	2.01
p93791	1,720,399	940,745	1.83

Table 1. STESOC compared to a bus-based test strategy for fixed-length scan chains

7 Future Work

Three major points have to be studied.

- The first one consists in designing a new TCPU internal architecture to improve test speed. We need also to synthesize the TCPU to evaluate the additional silicon area overhead induced by our approach.

SOC	STESOC	TR-Architect	$\frac{STESOC}{TR-A}$
d695	38,305	21,503	1.78
g1023	32,685	16,795	1.95
p22810	442,827	223,368	1.98
p34392	939,470	505,783	1.86
p93791	1,634,133	914,456	1.79

Table 2. STESOC compared to a bus-based test strategy for flexible-length scan chains

- Provide a solution to test the TCPU (BIST or software).
- Last but not least, is to tool up the TCPU with an LFSR in order to deliver pseudo-random patterns to IP cores. HTC test programs size can be reduced, and with the add of some extra deterministic test patterns, high fault coverage can be reach. Thus, those test programs can be stored in internal memory and the use of an external RAM can be avoided.

8 Conclusion

In software-based test of SoC, testing strategies for application specific cores are often somewhat ad-hoc. This paper has presented a software-based Test Access Mechanism to test those remaining components. Since this strategy is compliant with standards such as IEEE 1500, its integration in the test design flow is very straightforward. This software-based TAM is a dedicated test co-processor controlled by the embedded general purpose processor. Compared to an optimized bus-based TAM, STESOC requires two to three times more cycle to achieve the test. However, the test program is executed on-chip at SoC speed. The test process is not bound to the tester frequency and accuracy, allowing the use of low cost ATEs.

References

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

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

[3] 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.

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

[5] 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.

[6] 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.

[7] L. Chen and S. Dey. Software-Based Self Testing Methodology for Processor Cores. *IEEE Transactions on Computer-Aided Design*, March 2001.

[8] 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.

[9] Li Chen, Xiaoliang Bai, and Sujit Dey. Testing for interconnect crosstalk defects using on-chip embedded processor cores. In *Proceedings of the 38th conference on Design automation*, pages 317–320. ACM Press, 2001.

[10] 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.

[11] A.J. van de Goor, I.B.S. Tlili. March Tests for Word-Oriented Memories. In *Proceedings Design, Automation, and Test in Europe (DATE)*, page 501, 1998.

[12] S.M Fakhraie M.H. Tehranipour, Z. Navabi. An efficient BIST method for testing of embedded SRAMs. In *Proceedings International Symposium on Circuits and Systems (ISCAS)*, volume 5, pages 73–76, May 2001.

[13] Rochit Rajsuman. Testing a System-on-a-Chip with Embedded Microprocessor. In *Proceedings IEEE International Test Conference (ITC)*, pages 499–508, Atlantic City, NJ, September 1999.

[14] 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.

[15] S.M Fakhraie M.H. Tehranipour, Z. Navabi. Systematic test program generation for SoC testing using embedded processor. In *Proceedings International Symposium on Circuits and Systems (ISCAS)*, volume 5, pages 541–544, May 2003.

[16] 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.

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