T-Proc: An Embedded IEEE1500-Wrapped Cores Tester

Matthieu Tuna
University of Pierre et Marie Curie
LIP6 / ASIM
France, Paris
Email: matthieu.tuna@lip6.fr
Telephone: +33 (0)1 44 27 65 28

Mounir Benabdenbi
University of Pierre et Marie Curie
LIP6 / ASIM
France, Paris
Email: mounir.benabdenbi@lip6.fr

Alain Greiner
University of Pierre et Marie Curie
LIP6 / ASIM
France, Paris

Abstract— **This paper presents a software-based approach for testing IEEE1500-compliant SoCs. In the proposed approach, the test program is no more executed by the external-traditional tester but by the SoC itself. The novel feature is the use of a dedicated test processor called T-Proc embedded onto the SoC to test the components. Under the control of the embedded SoC microprocessor, the test processor executes the test programs stored in the outside external memory, through a functional embedded external RAM controller interface. Using the ITC02 SoC benchmarks a comparison is done between T-Proc and a classical bus-based test strategy.**

I. 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. 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 increases test application time, and huge memories are required in order to store test data. Moreover at-speed testing is becoming a very challenging task since the gap between the tester's frequencies and SoC frequencies is getting larger. Automated Test Equipments (ATE) are becoming more and more expensive since these trends are getting worse as circuits size shrinks and density increases. To deal with these limitations, a natural way consists in transferring some of the ATE capabilities into the SoC.

A solution to alleviate the role of the 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 Self-Test methodology is emerging. Software-Based Self-Test (SBST) consists on 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[1][2][3], system interconnects [4][5] and memories[6][7] have been well studied and convenient SBST solutions are provide. However, there is a lack of proposed solutions to test other kind of cores.

In the other hand, to answer the needs for a uniform test architecture for cores provided by different IP vendors, several working groups have been spawned [8], [9]. They define new standards in order to simplify test integration. The IEEE1500 working group defines the way a core must be shipped to be tested when integrated in a SoC. This group specified a Wrapper [10] 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 context of IP reuse and embedded test, we present a mechanism enabling the on-chip test of IEEE1500 wrapped cores. The test of such cores consists of, among others, delivering the test stimuli to the wrapped core interface, capturing the test response, and comparing this response with the expected one. This mechanism can be implemented using an embedded processor dedicated for the test purpose. This test processor is linked to the wrapped IP cores to be tested.

In the following, we present the T-Proc strategy: the targeted SoCs and the test execution process. Then, the next section presents the first results. Using the ITC'02 SoC benchmarks [11], a comparative study in terms of test application time and test data volume is done with a classical bus-based TAM. The additional silicon area overhead induced by the test processor is also presented. The last section concludes the paper.

II. T-Proc methodology

In this section we first describe some hardware requirements, the SoC template enabling the on-chip application of test patterns to wrapped IP cores. Then, we will see how the test is performed follow by a quick overview of the T-Proc internal architecture.

A. Targeted SoCs

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

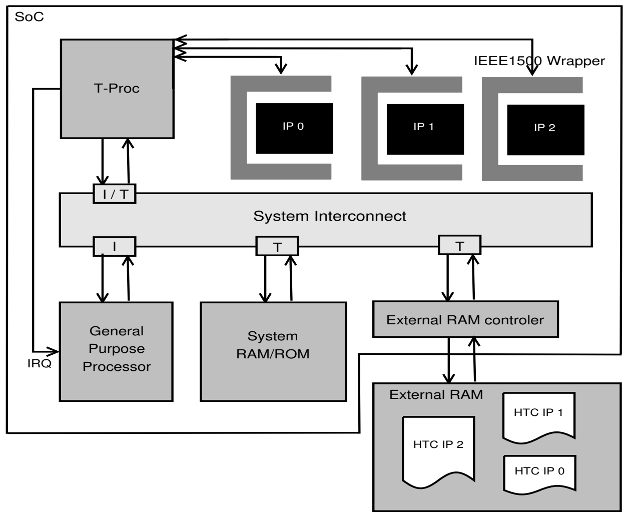

Fig. 1. T-Proc targeted SoCs

- 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. This 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. The test interface of each wrapped core includes test I/O data ports (one or many bits wide) and test control ports.

Besides this SoC scheme, we introduce a single new hardware component: a test processor entirely dedicated to SoC testing called T-Proc. 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 T-Proc can directly address the external RAM and thus, read test programs stored in. As a target, the T-Proc receives commands emitted by the GPP. The T-Proc can warn the GPP by the way of an Interrupt Request (IRQ).

B. Test execution

In this methodology, the test is performed by the GPP/T-Proc pair. T-Proc is in charge to process test programs. Each tested IP have its own test program, available in a format called HTC, specific to the T-Proc. Each HTC file is stored in the external memory (see figure 1). HTC test programs are fetched and executed by the T-Proc, each one contains scan data (test patterns) as well as test control informations (signal values to drive the scan shifting, the test pattern application and the response capture). Test responses are shifted out from the wrapped core into the T-Proc and compacted by the way of an internal Multiple Input Signature Register (MISR). T-Proc enables concurrent testing of many cores in order to minimize the test application time. During the test process, for each tested IP core, the T-Proc 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 T-Proc a start request containing the corresponding IP number. Thus, the T-Proc can start the test of the IP as described above. During the test process, the GPP can at any time send any request to the T-Proc 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 T-Proc. 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.

C. T-Proc: the test processor

The figure 2 shows the T-Proc internal architecture.

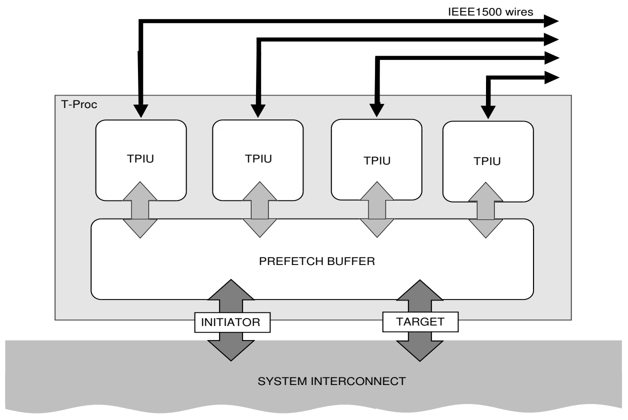

Fig. 2. T-Proc internal architecture

It includes two major kinds of components namely a Prefetch-Buffer (PB) and TPIUs (standing for Test Program Interpreter Unit). The Prefetch-Buffer acts like cache memory containing some fragments of the test programs. The PB is connected to the system interconnect and handles the T-Proc addressing system. The role defined for the PB is to provide an interface between the system interconnect and the TPIUs. The TPIU is the unit that executes the HTC program and convert it to a IEEE 1500 data stream. The number of TPIU is equal to the number of IPs to be tested. Each TPIU is dedicated to one wrapped IP core. For each added IP to be tested a dedicated TPIU must be plugged onto the Prefetch-Buffer. This modular architecture makes the surface of the T-Proc proportional to the number of tested IPs. Each TPIU is tooled up with a MISR in order to compact the test responses. This compaction reduces the total size of the HTC programs. An HTC program is a sequence 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. Software tools can automatically generated HTC test programs from CTL standard test pattern files. At this time, developed tools take a subset of STIL.

III. Experimental results

Five ITC'02 benchmarks [11] were used to show the test application time and the test data volume. The results are compared to those of a classical bus-based TAM strategy: TestRail optimized by the TR-Architect software toolbox[12]. The selected results for TR-Architect are those using a 32-bit channel width. The last paragraph of this section presents the area overhead introduced by the test processor.

A. *Test Application Time*

The 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 T-Proc model. An embedded RAM is added to store the MIPS binary, while an external RAM contains all the HTC test programs. Those latter components are connected via a VCI-compliant crossbar interconnect. For each module described in the benchmark a corresponding SystemC module is added in the plateform surrounded by an IEEE 1500 wrapper. The connections between the T-Proc 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 number and and scan-chains lengths are not designed yet and can be optimized. Thus, the module contains flexible-length scan-chains. Table I and II shows respectively the test application time (in terms of cycle number) for fixed-length scan-chains modules and 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, T-Proc requires only a factor less than 2 compared to TR-Architect. It is an expected result that this embedded 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 T-Proc is a "plug&play" device 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 presented 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 T-Proc. 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.

SOC	T-Proc	TR-Architect	$\frac{T-Proc}{TR-A}$
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 I
T-PROC COMPARED TO A BUS-BASED TEST STRATEGY FOR FIXED-LENGTH SCAN CHAINS

SOC	T-Proc	TR-Architect	$\frac{T-Proc}{TR-A}$
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 II
T-PROC COMPARED TO A BUS-BASED TEST STRATEGY FOR FLEXIBLE-LENGTH SCAN CHAINS

B. *Test Data Volume*

Each ITC'02 benchmark consists in a set of M modules. For each module $m \in M$ we have the number of functional inputs i_m, the number of functional outputs o_m, the number of functional bidirectional b_m, the number of scan-chains s_m, and for each scan-chain k the length of the scan-chain in flip-flops $l_{m,k}$. Thus, for each module m the total number of scan flip-flops is $f_m = \sum_{k=1}^{s_m} l_{m,k}$. And finally, for each module m we have the number of test patterns p_m. Since the size of a test stimuli corresponds to the internals scan-chains plus the functional inputs of the module m, the total size, in bits, of the test stimulus is $TS_m = (i_m + b_m + f_m) * p_m$. In the same way, test responses corresponds to the internals scan-chains plus the functional outputs, then, the total size of the test responses is $TR_m = (o_m + b_m + f_m) * p_m$. Thus, for each module m the total test data volume is $v_m = TS_m + TR_m$. For one benchmark b the total test data volume is $V_b = \sum_{m=1}^{M} v_m$. This volume is represented in the histogram (see figure 3) by the first bar of each benchmark called "raw test data". A test program of a classical bus-based strategy contains at least this volume of data plus a certain amount of test control. Since we do not have informations about this amount of test control we only consider this ”raw test data” volume in the following. For T-Proc, each HTC test program of each module consists in test stimulus as describe above and test controls interpreted by the TPIU. Test responses are shrank to only one 24-bits signature. Therefore, the second bar in the histogram represents the sum of all HTC test program in the case of fixed-length scan-chains. Finally the last bar represents the sum of all HTC test program in the case of the flexible-length scan-chains. The difference between the two latter cases, in favor of the flexible-length scan-chains architecture, is because in the fixed-length scan-chains architecture this number is not optimized for the T-Proc and results in padding bits increasing the test program size.

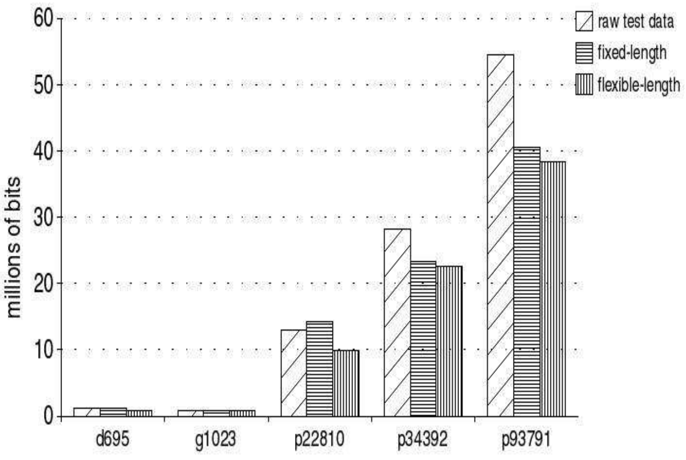

Fig. 3. Test Program Size

The Table III shows the percentage of the test program size reduction compared to the ”raw test data”. In average, the reduction for the fixed-length scan-chains architecture is about **11%** and for the flexible-length scan-chains the reduction is about **22%**.

	d695	**g1023**	**p22810**	**p34392**	**p93791**
fixed-length scan-chains	8.66%	17.09%	-10.50%	16.92%	25.57%
flexible-length scan-chains	24.93%	17.09%	23.01%	19.51%	29.65%

TABLE III

TEST DATA VOLUME REDUCTION COMPARED TO A BUS-BASED TEST STRATEGY

C. Area Overhead

A VHDL model of the test processor has been realized. Synopsys tools were used to synthesize the model. The model used for the layout generation contains a Prefetch-Buffer with four TPIUs as presented in the figure 2. Thus, it can test four different wrapped IP cores. The layout has been generated using the ALLIANCE CAD tools[13] with a $0.13\mu m$ standard cell library. The test processor surface is $0.45mm^2$. Since this test processor can test four different IP cores, the additional surface to test one IP is $0.11mm^2$. If we take into consideration the surface of an IP, for instance a MIPS with its cache memory has a surface equal to $2.5mm^2$ [14]. So the extra added surface for such core is only **4%**. This area overhead seems to be reasonnable since in BIST techniques an area overhead up to 10% is widely accepted.

IV. CONCLUSION

This paper has presented T-Proc a test purpose dedicated processor. Under the embedded general purpose processor control, it enables the on-chip test of wrapped IP cores. Since this strategy is compliant with standards such as IEEE 1500, its integration in the test design flow is very straightforward. Reusing the functional resources for test purpose minimizes the need of extra hardware dedicated for test. In traditional bus-based test strategies the tests of the main cores are merged in a single complex program, hard to develop and not easy to modify. T-Proc offers much more flexibility since each test program execution of each core is independent from the other. Moreover the compaction capability can reduce the test program size up to 30%. Compared to TR-Architect, T-Proc requires two to three times more cycles 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] F. Corno, M. S. Reorda, G. Squillero, and M. Violante, “On the Test of Microprocessor IP Cores,” in *Proceedings Design, Automation, and Test in Europe (DATE)*, Munich, Germany, Mar. 2001.

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

[3] 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, Mar. 2003.

[4] L. Chen, X. Bai, and S. Dey, “Testing for interconnect crosstalk defects using on-chip embedded processor cores,” in *Proceedings of the 38th conference on Design automation*. ACM Press, 2001, pp. 317–320.

[5] 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)*, Marina del Rey, CA, May 2001, pp. 204–209.

[6] S. F. M.H. Tehranipour, Z. Navabi, “An efficient BIST method for testing of embedded SRAMs,” in *Proceedings International Symposium on Circuits and Systems (ISCAS)*, vol. 5, May 2001, pp. 73–76.

[7] R. Rajsuman, “Testing a System-on-a-Chip with Embedded Microprocessor,” in *Proceedings IEEE International Test Conference (ITC)*, Atlantic City, NJ, Sept. 1999, pp. 499–508.

[8] “IEEE P1500 Web Site,” http://grouper.ieee.org/groups/1500/.

[9] “VSI Alliance Web Site,” http://www.vsi.org/.

[10] E. J. Marinissen *et al.*, “On IEEE P1500’s Standard for Embedded Core Test,” *Journal of Electronic Testing: Theory and Applications*, vol. 18, no. 4/5, pp. 365–383, Aug. 2002.

[11] E. J. Marinissen, V. Iyengar, and K. Chakrabarty, “ITC’02 SOC Test Benchmarks Web Site,” http://www.extra.research.philips.com/itc02socbenchm/.

[12] S. K. Goel and E. J. Marinissen, “Effective and Efficient Test Architecture Design for SOCs,” in *Proceedings IEEE International Test Conference (ITC)*, Baltimore, MD, Oct. 2002, pp. 529–538.

[13] “ALLIANCE CAD tools Web Site,” http://www-asim.lip6.fr/recherche/alliance/.

[14] “MIPS Technologies, Inc Web Site,” http://www.mips.com.