Reducing Test Time With Processor Reuse in Network-on-Chip Based Systems

Alexandre M. Amory, Érika Cota
Instituto de Informática-UFRGS
{amamory,erika}@inf.ufrgs.br

Marcelo Lubaszewski
Instituto de Informática-UFRGS
Dep.de Engenharia Elétrica-UFRGS
IMSE-CNM–Universidad de Sevilla
luba@{eletro.ufrgs.br, imse.cnm.es}

Fernando G. Moraes
Faculdade de Informática-PUCRS
moraes@inf.pucrs.br

ABSTRACT
This paper proposes a test planning method capable of reusing available processors as test sources and sinks, and the on-chip network as the access mechanism for the test of cores embedded into a system on chip. The resulting test time of the system is evaluated considering the number of reused processors, the number of external interfaces, and power dissipation. Experimental results for a set of industrial examples based on the ITC'02 benchmarks show that the cooperative use of both the on-chip network and the embedded processors can increase the test parallelism and reduce the test time.

Categories and Subject Descriptors
B.8.1[**Performance And Reliability**]: Reliability, Testing, and Fault-Tolerance

General Terms
Algorithms, Reliability.

Keywords
SoC test, core-based test, software-based test, computer-aided test (CAT), network-on-chip, NoC testing.

1. INTRODUCTION
As the number of cores embedded in the system increases, authors agree that a single broadcast or bus-based communication architecture can no longer meet the system requirements in terms of bandwidth, latency, and power consumption [3][18]. One emerging solution is the use of integrated switching networks, called networks-on-chip (NoC), to interconnect the cores in a system-on-chip (SoC) [2][7].

Assuming that a NoC is used as the communication platform for a system, the reuse of this structure as a test access mechanism (TAM) has been proposed in [5][6] and resulted in a cost-effective solution for the SoC testing, combining reduced test time with minimal test cost. However, results for the network reuse show that the test parallelism (and, consequently, the test time) is very dependent on the number of interfaces available during test.

SBCCI'04, September 7–11, 2004, Pernambuco, Brazil.

According to [8], there is an emerging trend pointing to a "sea of processors" with hundreds of heterogeneous processors connected by a network-on-chip. Considering this trend and the increasing complexity for testing such complex systems, the reuse of both, the processor and the on-chip network, seems an interesting approach to reduce test costs.

Software-based test of IP cores, i.e., the test based on the reuse of an embedded processor, presents several advantages, such as flexibility, requirement of fewer expensive ATEs, less test pins at the system interface, and minimal area overhead [13]. Current SoCs have an increasing number of processing units rather than complex custom blocks [8][15], which expands the possibility of reuse for testing purposes.

There are two important drawbacks for the processor reuse during test. Since the bus-based communication architecture is assumed [1][9][11][13], one must choose between the test serialization and the definition of dedicated test access mechanisms. The first option is less expensive in terms of hardware, but leads to larger test times. The second one reduces the test time at the expense of additional hardware. Thus, even if more than one processor is available within the chip, the possibility of reuse may be limited by the communication costs. The second drawback is related to the power consumption during test. Although the reuse of the embedded processor avoids the addition of new blocks in the system, thus reducing the number of active cores during test, the power consumption of the processor itself, while running the test program, may be quite high.

This paper evaluates the reuse of embedded processors when a network-on-chip is used as test access mechanism. The proposed approach tackles the main drawbacks of both the processor and the network reuse during test. On one hand, the network structure provides the means to increase test parallelism in a multiprocessor system. On the other hand, the presence of reusable processors provides embedded test sources and sinks, which reduces the requirements in terms of test interfaces and external tester capabilities. The new NoC-based and software-based approach is evaluated with respect to the system test time and considering power constraints. Experimental results are presented for three examples based on benchmarks of the ITC'02 SoC Test Benchmarks [16], and considering the reuse of two types of processors. Such experiments show that a significant test time reduction can be achieved when the reused processor requires fewer cycles to generate/analyze test data. The paper is organized as follows: Section 2 reviews some related work. Section 3 defines some details of the assumed software-based strategy. Section 4 presents the NoC model used in this work. Section 5 explains the combination of the processor reuse with the previously proposed NoC reuse strategy. The experimental setup is explained in Sec-

tion 6, whereas experimental results are presented in Section 7. Section 8 concludes the paper.

2. RELATED WORK

The reuse of an embedded processor for testing other cores in a system-on-chip has been largely studied and presented in the literature [9][10][11][13][14]. The approaches proposed so far consider different aspects of the processor reuse, ranging from the reuse as test controller [1] to the use as a built-in test pattern generator with compression features [11]. In all the previous works, a bus-based communication architecture is assumed.

Huang *et al.* [9] developed a bus-based architecture with a MIPS processor, a PCI bus and VCI interfaces. Using this architecture, they evaluated the test time and fault coverage of some ISCAS89 benchmarks.

Lai and Cheng [14] used the same architecture presented in [9] to evaluate test programs generated for four ISCAS89 benchmarks, using the DLX processor. The test program length ranges from 40 to 27,000 bytes while the test time ranges from 94 to 30,430 clock cycles. The results point to important requirements in terms of test memory and test time compared with hardware-based test.

Hwang and Abraham [10] developed a bus-based architecture with an ARM processor and Wishbone interface. The authors compared the test time and area overhead between software-based test and boundary scan. In both cases, the software-based test presented better results. In [11], the same authors evaluated a new test pattern compression method in which test data can be decoded rapidly on embedded processors. They compared the compression results using ISCAS89 benchmarks as case study with other compression methods applied to embedded processors.

Amory *et al.* [1] developed a CAD tool, which helps the designer to integrate cores on a bus-based SoC and generate test programs. These test programs are loaded into the processors to test other cores. Results are presented using a subset of ISCAS benchmarks as cores under test.

In [6] the reuse of the available NoC is proposed to reduce the test costs in terms of area and pins. Furthermore, a test scheduling algorithm is presented to efficiently use this new TAM architecture. In [5], the scheduling algorithm is improved to increase the network parallelism by allowing the use of different paths by the same core. In addition, power constraints are included in the test scheduling.

None of the previous test reuse approaches could use the computational power of multiple processors to reduce the system test time. In this work, the reuse of the NoC is combined with the reuse of embedded processors. It will be demonstrated in the experimental results that not only the number, but also the characteristics of the reused processors must be evaluated before defining the best reuse strategy.

3. EMBEDDED PROCESSOR REUSE

A processor can be used during test in different ways. It can run a test program that reads the compressed test data from a memory and sends it to the core under test (CUT), or it can work as a test pattern generator emulating a pseudo-random BIST logic.

In this work, two academic processors are being used in the experimental results to evaluate the impact of the software-based test in a NoC-based system. The first one is a stack-based processor called FemtoJava [12]. This processor executes a subset of JAVA bytecodes and was originally developed targeting embedded applications. The FemtoJava considers address and data buses of 16 bits. The second processor is called R8 [4] and implements a load-store architecture. The processor presents a low complexity control unit, 16 general-purpose registers, and address/data bus of 16 bits.

Two parameters must be defined before establishing a reuse strategy: the average number of cycles required to run the test program, and the average power consumption of the test processor when running the test program. These parameters are called "processor test latency" and "processor test power", respectively, in the sequel of this paper.

Table 1, in the next page, presents three different information: the test program characteristics such as programming language (column 2) and the size of the resulting object code (column 3); the test program profile in terms of execution latency (column 4) and power dissipation (column 5); and the test information related to the processor test, such as number of test patterns (column 6), number of scan chains (column 7), the largest scan chain (column 8), and the total number of data flits (size of the test packet) necessary to test the processors (column 9).

The test programs were simulated to define the *test latency* of each processor. The *test power* consumption has been characterized considering the dynamic consumption during the execution of a test program. To evaluate the dynamic consumption, the processors are initially synthesized to an ASIC technology library for which power consumption of technology cells is available. The resulting netlist description is simulated reusing the same testbench used to validate the original RTL description. The testbench loads the test program to be evaluated. During this simulation, the switching activity of each cell of the technology library is captured. Then, the power consumption per clock cycle is computed by multiplying the number of toggles and the power consumption per switching. The total power consumption for the whole simulation is given by the sum of the power consumption in each clock cycle. As the test scheduling algorithm proposed in Section 5 considers the power consumption per cycle, the processor test power is defined as the average power consumption per clock cycle.

In addition, it is very important to take into account the test of the processor itself in the overall reuse strategy, since this core is usually one of the most complex and time-consuming blocks in the system with respect to testing. Table 1 presents the number of test patterns and the maximal length of the scan chains for each processor. This information determines, respectively, the number of test packets and the size of each packet to test the processors. The FemtoJava has more packets of smaller sizes compared to R8. The total number of flits, calculated by multiplying the number of packets by the size of each packet, necessary to test FemtoJava is 2,778 and for R8 is 4,303.

Table 1 - Test processors parameters.

processor	test program		profile			processor test		
	language	size (bytes)	test latency (cycles)	test power (mW)	# of test patterns	# of scan chains	scan chain length	# of flits
femtojava	Java	25	140	0.17714	463	30	6	2778
R8	Assembly	24	57	0.07500	331	30	13	4303

4. USING THE NOC DURING TEST

NoCs typically use the message-passing communication model, and the processing cores attached to the network communicate by sending and receiving request and response messages. To be routed by the network, a message is composed of a header, a payload, and a trailer. The header and the trailer frame the packet, and the payload carries the data being transferred. The header also carries the information needed to establish the path between the sender and the receiver. Depending on the network implementation, messages can be split into smaller structures named packets, which have the same format of a message and are individually routed. Packet-based networks present better resource utilization, because packets are shorter and reserve a smaller number of channels during their transfer. In this work, a packet-switched network model named SOCIN (System-on-Chip Interconnection Network) is used in the experiments. Details of SOCIN architecture can be found in [17].

To reuse the available on-chip network as an access mechanism, the test vectors and test responses of each core are firstly expressed as a set of packets to be transmitted through the network. The static schedule of the test packets ensures the maximum utilization of the network resources that can be achieved through a test interface. This means that some channels and routers remain unused during test, not because of the traffic, but because the number of test interfaces is small. In other words, if more interfaces with a test controller were available, the network usage and the test parallelism could be increased, thus reducing the system test time.

5. NOC AND PROCESSOR REUSE

To combine the reuse of both, the processor and the NoC during the system test, we have to include the processor as another possible test interface in the NoC reuse approach proposed in [5].

Although the inclusion of the new test interface is straightforward, one must ensure that the resulting test scheduling meets the following restrictions:

1. The processor must be tested by an external interface before being used to test another core in the system;
2. When a core uses the processor as test source/sink, all packets to/from that core must use the same path. If this restriction is not set, a core may have some packets coming from the external tester and other packets coming from the internal test processor. This complicates the test controlling and would require more expensive wrappers;
3. A test processor cannot be used by more than one core at the same time. Again, this processor sharing requires that different test programs are concurrently loaded and executed by the processor, requiring additional test time to switch the context.

The original power-aware test scheduling algorithm presented in [5] was modified so that the restrictions of the software-based test are met. In addition to the external interfaces, embedded processors can be listed as test interfaces as well. For such processors, one must provide the associated latency and power information required to generate or analyze the test packet. External interfaces have zero latency and zero power consumption for transmitting a packet. The embedded IPs listed as test interfaces are given priority in the test scheduling. They are listed as the first ones to be tested so that they can be further used by other cores. The embedded reusable processors are tested by the external tester. When the test of a reusable processor finishes, this resource becomes available as another test interface to be used by other cores.

Equation 1 defines the time in number of cycles required to transmit a packet through a path in the SOCIN network, according to the test interface used. *Latency* is the number of cycles required to generate or analyze the test data. For an external interface, *Latency* = 0. However, if the interface is a processor, the latency assumed is the one presented in Table 1. T_{router} indicates the number of cycles spent by the packet header in each router to establish the path; $Nb_{routers}$ is the number of routers in the path; $T_{headers}$ indicates the number of cycles required to pack and unpack a header; *payload* is the number of flits with useful data in the packet. For the SOCIN network, $T_{router} = 3$, $T_{header} = 1$, and a test header of one flit is assumed, in addition to the packet header originally present in each message. For each packet, two extra cycles are assumed so that the core can process the test and the wrapper is ready to pack and deliver the response packet.

$$T_{packet} = Latency + T_{header} + 2 + T_{router} * Nb_{routers} + payload \quad (1)$$

5.1. Power-aware test scheduling algorithm

The main advantage of the network reuse during test is the possibility of parallelism provided by this communication platform. However, as more cores are tested in parallel, the system power consumption during test may become an issue. Therefore, the scheduling algorithm must also consider power consumption.

In the proposed test technique, there are five sources of power consumption: the core, the wrapper, the router, the communication channel, and the test interface if a processor is reused. The dynamic consumption per cycle for each source is calculated as a function of the number of active gates, using the equations presented in [5].

The scheduling can be modeled as a resource-constrained problem, since there are a limited number of resources (paths within the network) that can be used by the packets. Each packet of each core must be scheduled to ensure that a response for each test vector is delivered. The goal is to minimize the total execution time of the tasks, while maximizing the utilization of the avail-

able resources. The algorithm processes the available time slots in increasing order (starting at zero) and schedules as many operations as possible at a certain slot before moving to the next. A *ready-list* L_t contains the packets that can be scheduled from a given instant of time, and the algorithm will associate each packet to an available path for a certain time interval. Initially, L_t contains all packets carrying the first test vector of each core.

The cores that can be reused as test controllers have priority to be tested. Then, all other cores are sorted in decreasing order of test time. On the other hand, the access paths in each direction for each core are sorted in increasing order of length (number of routers). Thus, the scheduling algorithm tries to associate the shortest path to the most expensive core to minimize the global test time.

The power consumption per cycle is considered in the scheduling by assigning this information to each time slot of the test schedule. For each slot, the total power consumption is calculated as the sum of the power consumed by each packet scheduled to that slot. The packet consumption, on its turn, is the sum of the network consumption to transmit the packet (which depends on the number of routers and channels used by the packet) and the core consumption to receive (unpack) and process the test data. The system power limit P_{max} must be respected at each time slot s, that is, $Total_{power}(s) < P_{max.}$ Thus, before scheduling any packet, the total power required to transmit this packet is calculated. If the addition of this value to the total power consumption of the slot does not exceed the power consumption limit P_{max} defined for the system, that packet can be scheduled. Otherwise, the packet is scheduled later.

6. EXPERIMENTAL SETUP

Experimental results for the proposed reuse approach are presented for three examples based on the benchmarks d695, p22810, and p93791 of the ITC'02 SoC Test Benchmarks set [16]. The number of cores of d695, p22810, and p93791 are, 10, 28 and 32, respectively. The use of the original benchmarks is not possible because the functionality of the cores is not available and the definition of any core as an embedded test processor may lead to misleading results since the test time, the test power consumption and the test latency of the chosen core can not be estimated accurately.

As presented in Table 1, R8 has lower latency and power consumption than FemtoJava, but FemtoJava is tested faster than R8. These processors have been chosen to expose the benefits and the limitations of the proposed test method.

Let us define new systems composed of all the cores described in the original benchmarks and additional FemtoJava or R8 processor cores. For d695 system, six cores are added, whereas for p22810 and p93791 benchmarks eight cores are added. The total number of cores of the new systems is 16, 36, and 40, respectively. The network dimensions for each system are, respectively, 4x4, 5x6 and 5x5. Hereafter, d695_R8 indicates the system composed by the original benchmark and six R8 processors, while d695_FJ indicates the system with the original benchmark plus six FemtoJava processors. Similarly, let us define systems p22810_R8, p22810_FJ, p93791_R8, and p93791_FJ with eight processors of each type added to the original systems. For each system, a random placement for all cores within the NoC is defined and the same placement is used in all experiments. This means that the only difference between d695_R8 and d695_FJ is the added processors, and the same applies for the other systems. The test time of the new NoC-based systems is evaluated in the next section.

7. EXPERIMENTAL RESULTS

We evaluated the resulting test time for the systems considering: (*i*) increasing number of reusable processors; (*ii*) increasing number of external interfaces.

7.1. Test time versus number of reused processors

Figure 1 presents the test time results for systems d695_R8, p22810_R8, and p93791_R8, when different numbers of R8 processors are reused for test and two external interfaces (one input and one output) are used. Experiments with and without power constraint are presented for each system. This constraint is defined as a percentage of the sum of the power consumption of all cores. Thus, for example, a power limit of 50% indicates that the power limit corresponds to half of the sum of the power consumption of all cores in test mode. Notice that in a real case, the designer can define any power limit.

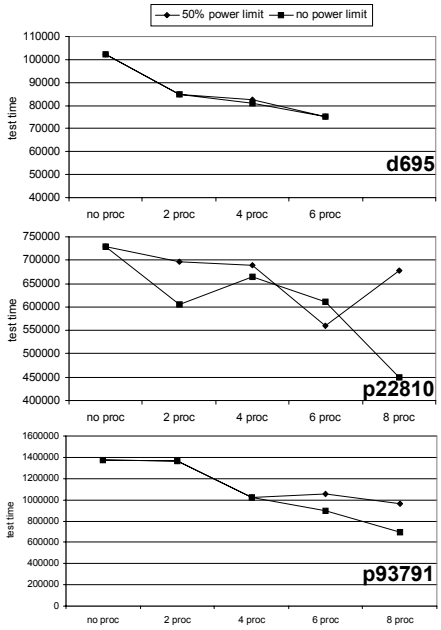

Figure 1 - Test times with R8 processors.

Table 2 presents the system test time reduction percentage when R8 processors are reused. The table compares the system test time with no processor reuse with the minimal and maximal test time presented in Figure 1, when there is processor reuse.

Table 2 – System test time reduction percentage using R8 processors.

		d695_R8	p22810_R8	p93791_R8
50% power constraint	**min**	+16.9	+4.3	+0.7
	max	+26.6	+23.2	+29.9
no power constraint	**min**	+16.9	+8.8	+0.7
	max	+26.6	+38.2	+49.3

The results presented in Figure 1 demonstrate that increasing the number of processors reused for test in most cases reduces the test time. One can observe that even smaller systems like d695_R8 can take advantage of the extra test interface. For this system, a test time reduction of up to 26% was achieved. For larger systems, as p22810_R8, the gain in the test time can be as high as 38%. For this system, one can also observe that the decreasing test time is not regular. When more processors are available the initial delay caused by the test of the processors is compensated by their reuse afterwards. Notice that larger systems imply networks with more channels and routers that can support the higher traffic of test packets when the processors are reused.

Table 3 presents the number of cores tested by the R8 processors for the solutions presented in Figure 1 considering power constraints. The first column presents the number of R8 processors available to be used during testing, varying from two to eight. Half of the processors are used as test input and the other half for test output. The remaining columns present the number of cores tested by the R8 processors. Consider the d695_R8 benchmark with two processors reused for test. The number *3/2* means that *3* cores use the R8 as test source and *2* cores use this block as test sink. The external tester is used as test source or sink by all cores that do not use the processor in that direction.

Table 3 – Number of cores tested by the R8 processors considering power constraints.

Processors (in/out)	d695_R8 # cores 16	p22810_R8 # cores 36	p93791_R8 # cores 40
1/1	3/2	2/9	2/1
2/2	4/4	3/7	12/14
3/3	7/7	6/7	16/14
4/4	-	7/13	7/8

For the p93791_R8 system, the power constraint limited the test parallelism. It can be observed in Table 3 that for 8 processors, the number of reused cores in p93791_R8 is reduced, since the power consumption for testing becomes prohibitive.

Figure 2 presents the evaluation of test time versus number of reused processors for systems d695_FJ, p22810_FJ, and p93791_FJ, whereas Table 4 presents the percentage of test time reduction of Figure 2 when the reused approach is compared to the purely external test.

Table 4 – System test time reduction percentage using FemtoJava processors.

		d695_FJ	p22810_FJ	p93791_FJ
50% power constraint	**min**	-13.62	-76.08	-24.47
	max	+14.43	+4.56	+14.06
No power constraint	**min**	-13.61	-195.78	-10.82
	max	-2.84	+9.55	+21.45

As presented in Table 1, the test latency and power of the FemtoJava IP are higher than those of R8, although this processor takes fewer cycles to be tested. One can observe in Figure 2 that the high latency and power consumption of the FemtoJava processor precludes its reutilization as test resource, and the test priority and serialization imposed by the possibility of reuse may increase the system test time.

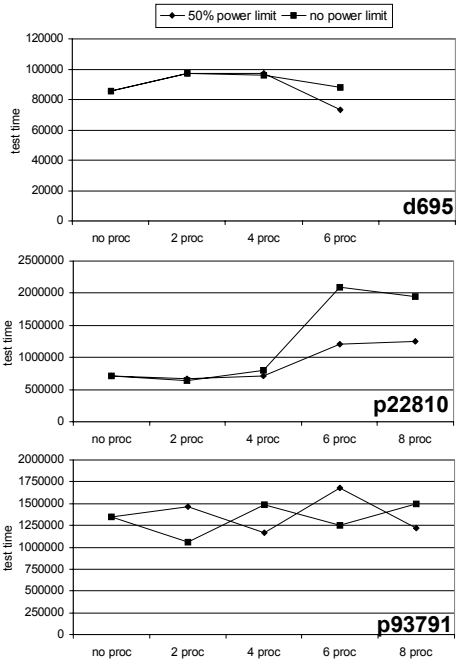

Figure 2 - Test times with FemtoJava processors.

7.2. Test time versus number of reused processors and external interfaces

Considering only the R8 processors, Table 5 presents the resulting test time when the number of external interfaces increases. These results aim at identifying the point where the reuse of embedded processors is not worth compared to the reuse of an external interface. The first column presents the number of external interfaces, the second column presents the availability of reusable processors, and the remaining columns present the test time for the systems. As it can be seen in Table 5, the test time with reuse is reduced (see *minus* signal) when there is only one external interface. But, increasing the external interfaces, the test time with reuse can be increased (see *plus* signal) or no processor is assigned to the test (see *equal* signal).

Table 5 – Test time as function of external interfaces, with R8 processors and 50% power constraint.

External interface (in/out)	Processor (in/out)	d695_R8	p22810_R8	p93791_R8
1/1	**0/0**	102264	728240	1374984
	1/1	84952-	696890-	1370807-
2/2	**0/0**	59225	455585	885096
	1/1	55093-	455585=	1119032+
3/3	**0/0**	40709	362066	725253
	1/1	41037+	362066=	725253=
4/4	**0/0**	-	365151	604543
	1/1	-	365151=	604543=

(-) the reuse of processors reduced the test time
(+) the reuse of processors increased the test time
(=) no processor is assigned to the test

8. FINAL REMARKS

This paper evaluated the impact of reusing processors to test a NoC-based system. The processors available in the system are programmed with a test application to test some cores of the system. This approach increases the number of test sources/sinks to explore the NoC parallelism reducing the system test time.

Results presented demonstrate the effectiveness of this approach to reduce the system test time when the reused processors have a small latency for test generation and analysis, but also presents its limitations.

The first limitation observed in Section 7.1 demonstrates that, when the processor has high latency and power cost to be reused for test, it may not decrease the system test time. The second limitation demonstrates that the proposed approach is interesting when there are an insufficient number of external interfaces, since the NoC parallelism cannot be fully explored. However, Section 7.2 demonstrated that there is no advantage of reusing processors when there are a significant number of external interfaces, since the parallelism is already fulfilled by the external interfaces.

We are presently working on the definition of a test scheduling algorithm capable of tackling the reuse of processors with higher latency and power consumption. Furthermore, we are also considering the evaluation of this technique for systems with more external interfaces.

9. ACKNOWLEDGMENTS

This work was partially supported by the Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq), under scholarship grant 307655/2003-2, Projeto Nacional de Microeletrônica (PNM) and Projeto Integrado de Pesquisa, under grant 524327/96-3.

10. REFERENCES

[1] Amory, A.M.; Oliveira, L.A. and Moraes, F.G. Software-Based Test for Non-Programmable Cores in Bus-Based System-on-Chip Architectures. In *VLSI-SOC*, 2003, 174-179

[2] Benini, L. and De Micheli, G. Networks on Chips: a New SoC Paradigm. *IEEE Computer*, 35, 1, 2002, 70-78.

[3] Bolotin, E.; Cidon, I.; Ginosar, R. and Kolodny, A. Cost Considerations in Network on Chip. Special issue on Networks on Chip, *Integration - the VLSI journal*, 2003.

[4] Calazans, N. L. V.; Moraes, F. G.; Marcon, C. A. M. Teaching Computer Organization and Architecture with Hands-On Experience. *32nd ASEE/IEEE Frontiers in Education Conference*, November, 2002.

[5] Cota, E.F.; Carro, L.; Wagner, F. and Lubaszewski, M. Power-aware NoC Reuse on the Testing of Core-based Systems. In *International Test Conference*, 2003, 612-621.

[6] Cota, E.F.; Kreutz, M.E.; Zeferino, C.A.; Carro, L.; Lubaszewski, M. and Susin, A.A. The Impact of NoC Reuse on the Testing of Core-based Systems. In *IEEE VLSI Test Symposium*, 2003.

[7] Guerrier, P. and Greiner, A. A Generic Architecture for on-Chip Packet-Switched Interconnections. In *Design, Automation and Test in Europe Conference*, 2000, 250-256.

[8] Henkel, J., Closing the SoC Design Gap. *IEEE Computer*, vol. 36-6, 2003, 119-121.

[9] Huang, J.-R.; Iyer, M.K. and Cheng, K.-T. A Self-Test Methodology for IP Cores in Bus-Based Programmable SoCs. In *IEEE VLSI Test Symposium*, 2001, 198-203.

[10] Hwang, S. and Abraham, J.A. Reuse of Addressable System Bus for SOC Testing. In *ASIC/SOC Conference*, 2001, pp 215-219.

[11] Hwang, S. and Abraham, J.A., Test Data Compression and Test Time Reduction Using an Embedded Microprocessor. *IEEE Transactions on Very Large Scale Integration Systems*, vol. 11-5, 2003, 853-862.

[12] Ito, S. A.; Carro, L.; Jacobi, R. P. System Design Based on Single Language and Single-Chip Java ASIP Microcontroller. In *Design, Automation and Test in Europe Conference*, 2000.

[13] Krstic, A.; Lai, W.C.; Cheng, K.T.; Chen, L.; Dey, S. Embedded Software-Based Self-Test for Programmable Core-Based Designs. *IEEE Design and Test of Computers*, vol: 19-4, 2002, 18–27.

[14] Lai, W. C.; Cheng, K. T. Instruction-Level DFT for Testing Processor and IP Cores in System-on-a-Chip. In *Design Automation Conference*, 2001, 59 –64.

[15] Magarshack, P. and Paulin, P.G. System-on-Chip Beyond the Nanometer Wall. In *Design Automation Conference*, 2003, 419-424.

[16] Marinissen, E.J.; Iyengar, V. and Chakrabarty, K. A Set of Benchmarks for Modular Testing of SoCs. In *International Test Conference*, 2002, 519-528.

[17] Zeferino, C.A. and Susin, A.A. SoCIN: A Parametric and Scalable Network-on-Chip. In *Symposium on Integrated Circuits and Systems Design*, 2003, 121-126.

[18] Zeferino, C.A.; Kreutz, M.E.; Carro, L. and Susin, A.A. A Study on Communication Issues for Systems-on-Chip. In *Symposium on Integrated Circuits and Systems Design*, 2002, 121-126.