Testing TAPed Cores and Wrapped Cores With The Same Test Access Mechanism*

Mounir Benabdenbi† Walid Maroufi‡ Meryem Marzouki
LIP6 Laboratory
Couloir 55-65, 4 Place Jussieu, 75252 Paris Cedex 05, France
Tel. (+33)1 44 27 39 67 - Fax. (+33)1 44 27 72 80
Mounir.Benabdenbi@lip6.fr Meryem.Marzouki@lip6.fr

Abstract

This paper describes a way of testing both wrapped cores and TAPed cores within a System On a Chip (SoC) with the same Test Access Mechanism (TAM). The TAM's architecture, which is dynamically reconfigurable, scalable and flexible, is named CAS-BUS and have a central controller. All the cores can be tested this way in the same session through a modified Boundary Scan Test Access Port.

1 Introduction

Testing Systems on a Chip (SoCs) is one of the challenges of the new century. SoC test feature standards are currently under development [1], such as core wrappers (P1500 wrappers) and core test language (CTL). Given the increasing density of integration in the new chips, it becomes harder to have access, for test purpose, to the core I/Os because these cores are deeply embedded in the SoC. Thus defining new Test Access Mechanisms (TAMs) for SoCs becomes a common need. The TAM will not be standardized unlike the core wrapper or the test language. Defining a test access mechanism is then left to the SoC integrator depending on chip constraints.

A P1500 compliant reconfigurable TAM architecture named CAS-BUS has been presented in [2]. This architecture is controlled through IEEE 1149.1 features [3] and is both P1500 compliant (in its current status) at core level and 1149.1 compliant at SoC level.

Some TAMs, like the CAS-BUS TAM, are integrated in SoCs with wrapped cores, P1500 or proprietary wrappers [4], [5], [6]. Some others are used for testing TAPed cores [7], [8], [9]. But, to our knowledge, none of the existing TAMs simultaneously deals with wrapped cores and TAPed cores integrated in the same SoC. However, a system integrator may need to use in a SoC some TAPed cores with P1500 wrapped cores. These TAPed cores can be primarily designed as stand-alone chips. In order to keep the time to market of the SoC as short as possible, time cannot be spent on redesigning each TAPed core giving away the boundary scan features and replacing them with P1500 wrappers. On the other hand, a TAPed core cannot simply be wrapped on top of 1149.1 features, because of area overhead reasons as well as problems with the hierarchical global test control, which can lead to violations of the IEEE 1149.1 standard.

We present in this paper a way of testing SoCs including both wrapped cores and TAPed cores with a unique Test Access Mechanism. The CAS-BUS TAM, enhanced for TAPed cores testing needs, keeps its capabilities. It remains reconfigurable, scalable, flexible and compliant with the P1500 and 1149.1 standards. After a brief reminder of the architecture, the CAS-BUS TAM improvements will be described. The control of the overall architecture, including a modified boundary scan state machine, will then be presented. Before concluding, some benefits of this architecture will be discussed.

2 The CAS-BUS TAM

2.1 The CAS-BUS TAM architecture

The CAS-BUS (figure 2) is a TAM which main function is to provide access to embedded cores whatever the wrapper is. This TAM is made up of two main elements:

- **a bus** consisting of N wires,
- **a Core Access Switch (CAS)** (figure 1)

Each CAS selects from the N wires of the bus (e_i) the P ones that will be applied to the core wrapper inputs (o_i). It also connects the P outputs of the wrapper (i_i) to the CAS outputs (s_i). Unselected inputs (e_i) are bypassed to the out-

*This work has been partly supported by MEDEA-SMT Project
†Corresponding author
‡Now with Nortel Networks, Ottawa

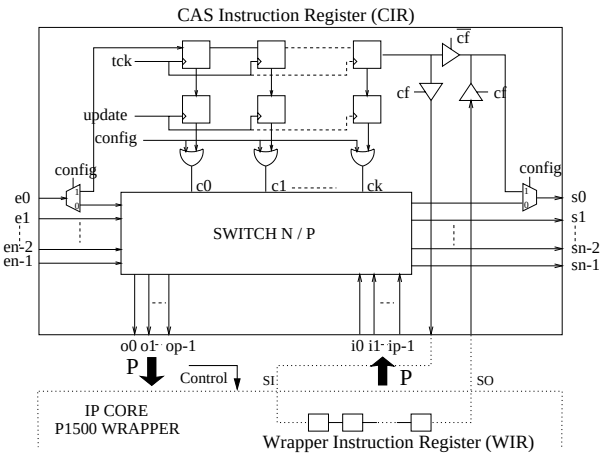

Figure 1. CAS architecture

puts (s_i). The CAS is composed of a Switch and a CAS Instruction Register (CIR). It is controlled by several signals. Signal c_f can connect the CIR to the Wrapper Instruction Register (WIR). Signals c_i control the Switch.

Controlled by the CIR, the Switch is in charge of routing the test stimuli and the test responses to/from the core wrapper. Two kinds of Switch implementation are possible: the Switch can be made up of N decoders (one per input wire) and some logic gates, but it can also be implemented with only one decoder and some logic gates. The Switch area and the length of the word control are different for the two implementations.

The decoder controls the multiplexing of the N/P I/Os. By selecting P wires among the N ones existing at the Switch inputs, the decoder must be able to address all the possible N/P combinations.

The number of combinations, as well as the Switch area, obviously depend on N and P values and on the chosen implementation. Results obtained for different values of N and P can be found in [3] and are presented in table 1.

This CAS-BUS TAM architecture is independent from any industrial proprietary SoC architecture. It can be used with any kind of wrappers and thus gives the SoC integrator more freedom for designing a SoC test architecture. The CAS-BUS is scalable, flexible and reconfigurable. More details on this architecture can be found in [2] [3].

2.2 The SoC Test control

The control of this TAM architecture must be simple without degrading the TAM advantages. Two options have been considered: defining an ad hoc test control mechanism or reusing a standard control architecture. The decision to reuse the IEEE 1149.1 Test Access Port (TAP) controller naturally came to manage our TAM functionalities. This solution fits well with our SoC test architecture, since it allows TAM control, wrapper control and usual Boundary Scan functionalities at the same time. The test control must allow to manage three mandatory test steps: core wrapper and CAS configuration, test vector application and test response analysis.

The normal Boundary Scan architecture has been extended with a CAS-Wrapper Coupling Register (CWCR), as shown in figure 2. The CWCR width is equal to the number of CASes. Each bit of this register corresponds to the value of the configuration signal cf for each CAS. In our modified TAP architecture, TDI and TDO is a bus. This TDI/TDO bus width may vary from 1 (normal Boundary Scan architecture) to N (width of the CAS bus).

Three new instructions are needed to control the three mandatory test steps. They have been defined as Boundary Scan optional instructions.

- the **CAS-Wrapper_COUPLING** instruction defines which CIR/WIR pairs must be connected in order to configure at the same time a core wrapper and the corresponding CAS. $TDI1$ and $TDO1$ are connected to the CWCR. Configuration values are shifted in the CWCR to indicate which WIRs will be connected to the CIRs.

- the **CAS_CONFIG** instruction configures all the CASes, together with the wrappers selected with the instruction CAS-Wrapper_COUPLING. The $config$ signals provide access to the CIR of each CAS. $TDI1$ and $TDO1$ are respectively connected to first input of the first CAS and to first output of the last one. The CIR and the WIR are then loaded with the appropriate values, putting each CAS in the correct scheme and each wrapper in the chosen mode.

- the **CAS_TEST** instruction, when loaded, allows the test vectors to be applied to the different cores and the test stimuli to be propagated to the SoC test outputs. All $TDIs$ and $TDOs$ are connected to the I/Os of the CAS-BUS. The set of test vectors can be concurrently shifted in to the IP cores inputs and the responses shifted out from the IP outputs.

As already said, the TDI/TDO bus width may vary from 1 to N. This width can be chosen by the SoC integrator, depending on SoC pins availability. In case the TDI/TDO bus width should remain equal to one, a compression/expansion mechanism has been developed to solve the SoC I/O bandwidth problem [10], [11].

Using the IEEE 1149.1 control part with scalable TDI/TDO lets the CAS-BUS TAM keep its main benefits and adds to it flexibility.

The global architecture is also dynamically reconfigurable. By correctly configuring the CASes, the test programmer can choose during each test session which core scan chains must be serialized in order to optimize their total length. The goal is to have for each of the N wires the same test length. If not, when testing the SoC, don't care stimuli

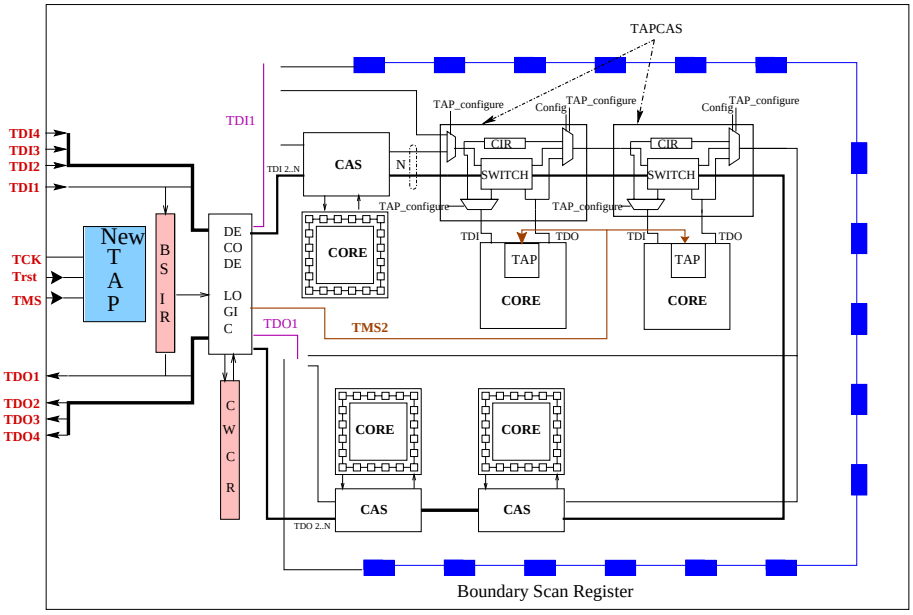

Figure 2. SoC Test architecture for TAPed and wrapped cores

must be applied at TDI inputs to complete the balance between scan chain lengths. Another advantage is that WIRs whose mode remains unchanged between two test configurations can be bypassed during the CAS_CONFIG phase. Using this reconfigurability aspect, test time can be saved.

The CAS-BUS architecture is simple, allows trade-off and provides a complete solution for testing SoCs. Its advantages over other approaches have been discussed in [3].

3 CAS-BUS architecture with TAPed cores

The test architecture presented above can be used when the cores within the SoC are wrapped cores. Since TAPed cores are not wrapped cores, they can only be tested through their I/Os (TMS, $TRST$, TDI, TDO and TCK). Unlike wrapped cores, these cores include a TAP controller. Obviously this needs to define a hierarchical test architecture able to drive these TAP controllers. Moreover, configuring the TAPed cores in the chosen mode cannot be done in the same way as wrapped cores. Direct access to the wrapper shift/update instruction register (WIR) can be enabled, while the instruction register within the TAPed core can only be accessed through the TAP finite state machine. Thus TAPed cores and wrapped cores cannot be configured serially in the same test step.

Our goal being to test both wrapped and TAPed cores within the same test step and with the same TAM, some improvements must be made on the CAS-BUS architecture in order to solve the problems induced by the TAP controllers.

The two main improvements consist of defining on the one hand a special Core Access Switch for TAPed cores and of modifying on the other hand the central TAP controller of the CAS-BUS architecture in order to manage the hierarchical aspect of the global control.

3.1 Core Access Switch for TAPed cores

Since TAPed cores cannot be serially configured with the wrappers, a new Boundary Scan instruction named **TAP_CONFIG** has been added, which enables the configuration of the TAPed Cores. The configuration of the wrappers is still made by the CAS_CONFIG instruction. Figure 2 shows a SoC example containing both kind of cores. This TAP_CONFIG instruction serially connects CASes of TAPed Cores (TAPCAS) one to the other and connects each TAPCAS to its associated core. It also connects $TDI1$ and $TDO1$ respectively to the first and the last TAPCAS.

The TAPCAS is slightly different from the normal CAS. The CIR and the Switch remain the same, the tristates disappear and multiplexers are added. However, two kinds of TAPCAS architectures are needed, depending on the placement of the TAPed core within the SoC. In the global chaining of all the CASes, if the TAPCAS follows a CAS then the TAPCAS must have a multiplexer at its inputs. In the other cases this multiplexer is not needed.

Obviously, for the TAPCAS, P is equal to 1: the TAPCAS is connected to TDI/TDO pair of the TAPed core. Depending on the boundary scan instruction, test data at TAPCAS inputs can be routed differently. The different multiplexers will route these data to:

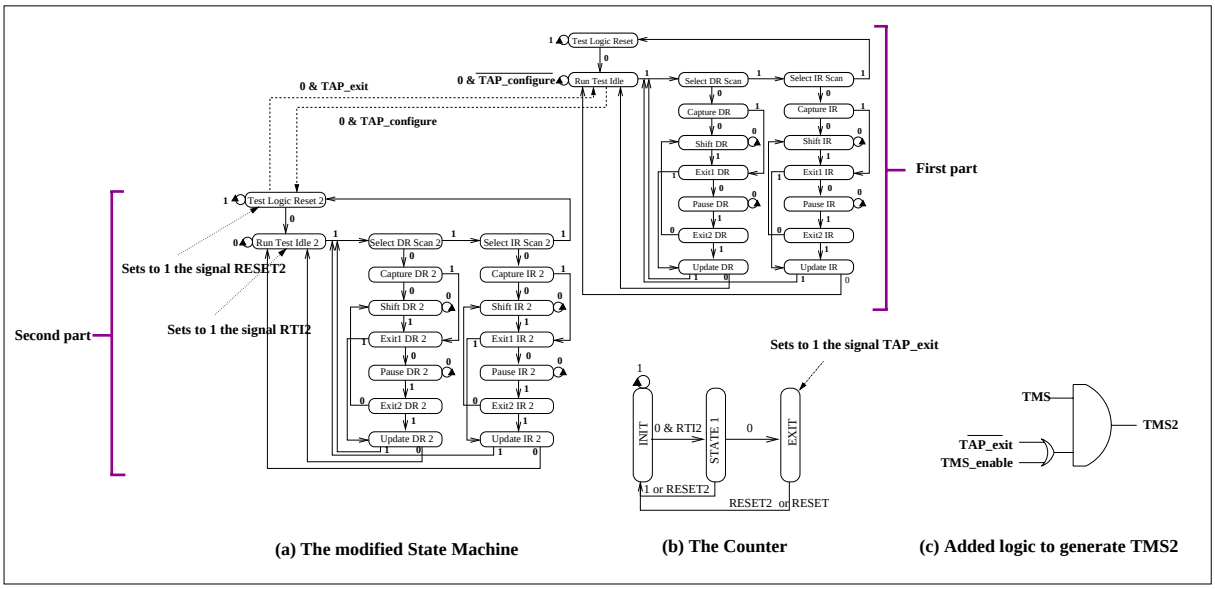

Figure 3. SoC Test architecture controller

- the TAPCAS instruction register (CIR). This is done with the CAS_CONFIG instruction. The switch is configured like all the other switches.

- the internal instruction registers of the TAPed cores. In that case the TAP_CONFIG instruction must be loaded. A configuration word is shifted in the internal instruction registers of the TAPed cores, these registers being daisy chained. The data skip the Switch in the TAPCAS and are directly applied to the TDI input of the TAPed core. The different operations like shifting, updating, etc., are executed thanks to the TMS2 signal that drives the internal TAP controller. This signal TMS2, described in the next section, is generated from the central TAP controller. Once the internal instruction registers are updated, the TAPed cores are then in the correct test mode (bypass, INTEST, etc.) and ready for test vectors application.

- the internal data registers of the TAPed cores. When the CAS_TEST instruction is loaded, test vectors are routed from one of the N wires of the TAPCAS inputs to the TDI input, through the Switch. The test vectors are shifted in the internal test data registers (boundary scan register, bypass register...). The responses to the stimuli are then shifted out to the next core or to the SoC outputs. Here also the shift/update operations are controlled through TMS2.

3.2 The New Central TAP Controller (CFSM)

The new central TAP controller (figure 3) is made up of a 1149.1 like TAP Finite State Machine (FSM) and a counter.

This Central FSM (CFSM) is in charge of controlling the global architecture as a classical boundary scan feature but must also drive the different Internal Finite State Machines (IFSM) of the TAPed cores. To do that, it was decided to add to the usual 16 states of a TAP, a second set of 16 new states corresponding to the states of the TAPed cores. Except Test logic 2 and Run Test Idle 2, the added states do not drive any signal and only are transition states. Depending on the instruction loaded, TMS will control the CFSM and the IFSMs or only the CFSM.

The test instructions to be loaded in the BS IR should respect the following chronological sequence:

- **CAS_WRAPPER_COUPLING** to define which CIRs and WIRs will be connected during the configuration step. This instruction operates only on CASes and wrapped cores.
- **CAS_CONFIG** to configure the CASes and the TAPCASes. The routing scheme of the overall Switches is defined.
- **TAP_CONFIG** to configure the test mode of the TAPed cores.
- **CAS_TEST** to apply test stimuli and shift out the test responses. The TAPed cores and the wrapped cores are tested concurrently through the same bus.

- TAP_CONFIG

This instruction is needed to shift values in the instruction registers of the TAPed cores.

When this instruction is loaded and updated, the signal TAP_configure is set to one. Leaving the Run Test Idle state (figure 3 (a)) we enter then in the second part of the CFSM in the Test Logic Reset 2 state. This state sets to 1 the

RESET2 signal that sets the counter (figure 3 (b)) in the state INIT. The counter is initialized, the signal TAP_exit is set to 0. TMS2 (figure 3 (c)) will then have the same value as TMS. To synchronize the CFSM and the IFSM, which states must be equivalent to those of the second part of the CFSM, TMS must be held at 1 during five cycles to reset the IFSM.

When entering in the Run Test Idle 2 state, the signal RTI2 is set to 1, Reset2 is set to 0 and if TMS is held at 0 during two cycles, the counter is put in the state EXIT. The TAP_exit signal is set to 1. This sets TMS2 to 0 and the IFSMs are no more controlled by TMS. The IFSMs are and stay in the Run Test Idle state. The counter is used not only to enable and disable IFSMs control but also to leave the second part of the CFSM.

If TMS is not held at 0 during two cycles, when the CFSM is in the Run Test Idle 2 state, the counter stays in the INIT state. The IFSMs are controlled by TMS and their current state is equivalent to the current state of the second part of the CFSM. The instruction words can be shifted in the TAPed core as they would be if placed in a classical boundary scan board.

- CAS_TEST

This instruction is used to shift in, apply and shift out test vectors to and from all the cores within the SoC. With the right configuration of the CASes and the TAPCASes, all the test data registers are connected as multiple scan chains. Internal test data registers of the TAPed cores (boundary scan or bypass registers) can be considered as internal scan chains of the P1500 wrapped cores. When updated this CAS_TEST instruction sets to 1 the signal TMS_enable and sets to 0 the signal TAP_configure. TMS controls the IFSMs through TMS2 (figure 3 (c)). During this test step the current state of the CFSM cannot be one of those of the second part.

When leaving the Update IR state (figure 3 (a)), the next state must be Run Test Idle. The configuration step of the TAPed cores having left the IFSMs in the Run Test Idle state, the synchronization of the CFSM and the IFSMs is done thanks to this state. By this way, the current state of all the finite state machines of the SoC being exactly the same, shifting and applying test vectors within wrapped and TAPed cores can be made at the same time.

The IFSMs remains controlled by the CFSM until a new instruction is loaded in the Boundary Scan Instruction Register (BS IR) of the SoC. This new instruction will set to 0 the signal TMS_enable. This is equivalent to setting TMS2 to 0 and the IFSMs are left in the state Run Test Idle. However, while shifting this new instruction in the BS IR, unknown values are concurrently shifted in the internal instruction registers of the TAPed cores. This is not important because the test responses have already been shifted out from the internal test data registers and because the next test session will reset these instruction registers. The next TAP configuration step will load these registers with the right values.

4 Benefits and Experimental Results

The main benefit of this new CAS-BUS architecture is that TAPed cores do not need to be tested apart from the other wrapped cores, using added test resources and hence increasing the test area overhead. TAPed cores can be tested with the CAS-BUS TAM taking advantage of the possible optimizations provided by this TAM. Test time can still be optimized through correct configuration of all CASes and TAPCASes.

With this architecture it is also possible to test only TAPed cores with the TAP_CONFIG instruction. The cores can be tested as stand-alone chips on a boundary scan board, controlled by the CFSM, because direct test access is provided to their I/Os, skipping the wrapped cores. However BIST testing can not be processed within these TAPed cores. During the TAP_CONFIG step the second part of the CFSM is active. If the RUN_BIST instruction is loaded in the TAPed core, to execute the self test, the current state must be Run Test Idle 2 during many cycles, with TMS held at 0. With this CFSM, this cannot be done. In that case, after TMS is held at 0 during two cycles, the next state is Run Test Idle, in the first part of the CFSM, and then the IFSMs stop being controlled by the CFSM. If really needed, a solution could be to add a register that would be concurrently loaded with the internal instruction registers of the TAPed cores. If the RUN_BIST instruction is detected in this new register, the current state remains in the Run test Idle 2 state without exiting to the main part of the CFSM.

As for the previous CAS-BUS architecture, interconnect testing can be performed, between wrapped cores and TAPed cores, assuming that the EXTEST instructions are loaded.

The main features of the new TAM have been synthesized and simulated using the Synopsys tools. Some results are presented in table 1. CAS (a) corresponds to a CAS which Switch is implemented by N decoders and CAS (b) a CAS with a Switch including only one decoder. The CAS Instruction Register width (k) and the area in terms of transistors (tr) are presented. A trade-off must be made between CAS (a) and CAS (b) depending on the area constraints and the scan cycles overhead needed for CAS configuration. For high values of N and P (great number of combinations) CAS (b) cannot be used because the area grows quickly with N and P. However TAPCASes (b) are more interesting than TAPCASes (a) since they are smaller and need fewer bits control. The area overhead induced by this CAS-BUS architecture is not significant since the TAP-

CASes are very small and the new TAP controller is implemented with 1264 transistors. However configuring TAPed cores, in comparison with wrapped cores, needs 27 extra scan cycles (loading the TAP_CONFIG instruction, entering and leaving the second part of the central controller).

This new CAS-BUS TAM architecture does not degrade the characteristics of the previous architecture but offers a solution for testing SoCs with TAPed cores. If the SoC does not have any TAPed cores included, the integrator can use the previous CAS-BUS architecture.

		CAS (a)		CAS (b)		TAPCAS (a)		TAPCAS (b)	
N	P	k	tr	k	tr	k	tr	k	tr
5	1	5	696	3	626	5	684	3	614
5	3	10	1.224	6	2.026	–	–	–	–
5	4	15	1.636	7	2.634	–	–	–	–
8	1	8	1.084	4	912	8	1.072	4	900
8	3	16	1.952	9	7.286	–	–	–	–
8	4	24	2.548	11	39.990	–	–	–	–
8	6	24	2.948	15	>100.000	–	–	–	–
10	1	10	1.338	4	1.044	10	1.326	4	1.032
10	4	30	3.236	13	>100.000	–	–	–	–

Table 1. CAS synthesis results

5 Conclusion

The architecture presented in this paper offers a complete solution for testing both wrapped and TAPed cores within a SoC, concurrently if needed. With the same test access mechanism, wrapped and TAPed cores can be tested, thanks to a hierarchical test control. In order to manage the TAPed cores testing, new CASes have been designed and the central TAP controller has been modified. This upgraded CAS-BUS TAM remains flexible, scalable and dynamically reconfigurable. It allows multiple trade-off regarding the choice of $Nmax$, the number of TDI/TDO couples, the kinds of CASes implementation and the CASes configurations. The area overhead induced by the TAPed core testing is not significant. The control of the global architecture is easy through a simple test access port. The architecture is both compatible with the direction of P1500 (as published so far) at core level and 1149.1 compliant at SoC level. The SoC integrator has the choice of using this architecture or the previous CAS-BUS TAM depending on the presence of TAPed cores in its SoC.

References

[1] E.J. Marinissen, Y. Zorian, R. Kapur, T.Taylor, and L.Whetsel. Towards a standard for embedded core test: An example. In *IEEE International Test Conference (ITC)*, pages 616–627, Atlantic City, NJ, September 1999.

[2] M. Benabdenbi, W. Maroufi, and M. Marzouki. Cas-bus: A scalable and reconfigurable test acces mechanism for systems on a chip. In *IEEE Design Automation and Test in Europe (DATE)*, pages 141–145, Paris, France, March 2000.

[3] W. Maroufi, M. Benabdenbi, and M. Marzouki. Controlling the cas-bus tam with ieee 1149.1 tap: A solution for systems on a chip testing. In *4th IEEE International Workshop on Testing Embedded Core-Based System-Chips*, pages 4.5.1–4.5.6, Montreal, Quebec, Canada, May 2000.

[4] E. J. Marinissen and al. A structured and scalable mechanism for test access to embedded reusable cores. In *International Test Conference*, Washington, DC, October 1998.

[5] L. Whetsel. Addressable test ports an approach to testing embedded cores. In *IEEE International Test Conference (ITC)*, pages 1055–1064, Atlantic City, NJ, September 1999.

[6] P. Varma and S. Bhatia. A structured test re-use methodology for core-based system chips. In *International Test Conference*, Washington, DC, October 1998.

[7] D. Bhattacharya. Hierarchical test acces architecture for embedded cores in an integrated circuit. In *IEEE VLSI Test Symposium (VTS)*, pages 8–14, Dana Point, CA, April 1998.

[8] B. Dervisoglu and J.Swamy. A novel approach for designing a hierarchical test access controller for embedded core designs in an soc environment. In *4th IEEE International Workshop on Testing Embedded Core-Based System-Chips*, pages 1.4.1–1.4.7, Montreal, Quebec, Canada, May 2000.

[9] L. Whetsel. An ieee 1149.1 based test acces architecture for ics with embedded cores. In *IEEE International Test Conference (ITC)*, pages 69–78, Washington, DC, November 1997.

[10] W. Maroufi. A new compression/decompression method for non correlated test patterns: Application to test pins expansion. In *IEEE European Test Workshop (ETW)*, Cascais, Portugal, May 2000.

[11] W. Maroufi, M. Benabdenbi, and M. Marzouki. Solving the i/o bandwith problem in system on a chip testing. In *XIII Symposium on Integrated Circuits and System design (SBCCI2000)*, Manaus, Brazil, September 2000.