TR-Architect: DfT and Test Support for SOC Designers

Sandeep Kumar Goel and Erik Jan Marinissen
Philips Research Laboratories,
Prof. Holstlaan 4, M/S WAY-41, 5656 AA Eindhoven, The Netherlands
Email: {SandeepKumar.Goel, Erik.Jan.Marinissen}@philips.com

Abstract—**This paper deals with the design of on-chip architectures for testing large system chips (SOCs) for manufacturing defects in a modular fashion. These test architectures consist of wrappers and Test Access Mechanisms (TAMs). We have developed a tool called TR-ARCHITECT for test architecture design. The tool efficiently determines the number of TAMs and their widths, the assignment of modules to TAMs, and the wrapper design per module, such that the SOC test length is minimized. In addition, the tool is also able to take into account constraints w.r.t (1) layout, (2) test control, (3) design hierarchy, and (4) user control. Using TR-ARCHITECT, we were able to successfully fit the test data volume of the Philips PNX8550 SOC onto its target ATE. Experimental results show that with the help of TR-ARCHITECT, further test length reductions up to 50% would have been possible for the same SOC.**

Keywords— **Core-based testing; core test wrapper; Test Access Mechanism (TAM); test architecture design; test scheduling**

I. INTRODUCTION

Modern semiconductor design methods and manufacturing technologies enable the creation of a complete system on one single die, the so-called *system chip* or SOC. Such system chips typically are very large ICs, consisting of millions of transistors, and containing a variety of hardware modules. In order to design these large and complex system chips in a timely manner and leverage external design expertise, increasingly reusable cores are utilized. *Cores* are pre-designed and pre-verified design modules, meant to be reused in multiple SOC designs [1]. Examples of cores are CPUs, DSPs, media co-processors, communication modules, memories, and mixed-signal modules.

Due to imperfections in their manufacturing process, all integrated circuits need to be individually tested for manufacturing defects. System chips are no exception to that rule. Modular test development is increasingly used for SOCs. Non-logic modules, such as embedded analog circuitry and memories require stand-alone testing due to their 'abnormal' circuit structure. Black-boxed third-party cores, such as *hard* (layout) cores and encrypted cores, for which no implementation details are known, need to be tested by the tests as supplied by their provider, and therefore also require stand-alone testing. But even for logic modules of which the implementation details are known, modular test development is an attractive alternative. Here, a modular 'divide-and-conquer' test development approach helps to reduce the test generation compute time and associated data volume. Finally, a modular test approach enables test reuse, which especially pays off if a core or module is used in multiple SOC designs.

In order to enable modular test development, an embedded module should be isolated from its surrounding circuitry and an electrical test access needs to be provided. Zorian et al. [2] introduced a generic conceptual test access architecture enabling modular testing of SOCs, consisting of three elements per module-under-test: (1) a test pattern *source* and *sink*, (2) a *test access mechanism* (TAM), and (3) a *wrapper*. The wrapper [3] is a thin shell around the module and can isolate the module from its surroundings. The wrapper also provides switching functionality between functional access to the module and test access through the TAM. The test architecture has a large impact both on the required vector memory depth per tester channel, as well as on the test application time of the SOC, two key parameters in the overall SOC test cost. In the remainder of this paper, we refer to these two parameters as 'test length'.

To design a test architecture for a given set of modules and a given number of test pins, an SOC integrator has to determine (1) the number of TAMs and (2) their widths, (3) the assignment of modules to TAMs, and (4) the wrapper design for every module. For a small SOC, having only a few modules and a few test pins, a good test architecture can be designed manually. However, the complexity of designing an architecture increases exponentially with the number of modules and test pins. Iyengar et al. [4] proved that the problem of designing an optimal test architecture is $\mathcal{NP}$ hard, indicating that the required compute time increases exponentially with the problem instance size. Therefore, there is a need for a tool which can efficiently search the solution space of feasible architectures and yield a (near-)optimal test architecture. We have developed such a tool, named TR-ARCHITECT. This paper presents a summary of the research work [5–12] carried out at Philips Research, Eindhoven in relation to the development of TR-ARCHITECT.

The sequel of this paper is organized as follows. Section II reviews prior work in this domain. Section III defines the problem of test architecture optimization, assuming the relevant parameters of modules and a maximal SOC TAM width are specified. Section IV describes our tool TR-ARCHITECT together with its various features. Section V presents the experimental results for the first industrial use of TR-ARCHITECT for the Philips PNX8550 SOC. Section VI concludes this paper.

II. PRIOR WORK

A. Test Architecture Design

Various test architectures have been described in literature. Aerts and Marinissen [13] described three scan-based test architectures depicted in Figure 1: (a) the *Multiplexing* Architecture, (b) the *Daisychain* Architecture, and (c) the *Distribution* Architecture.

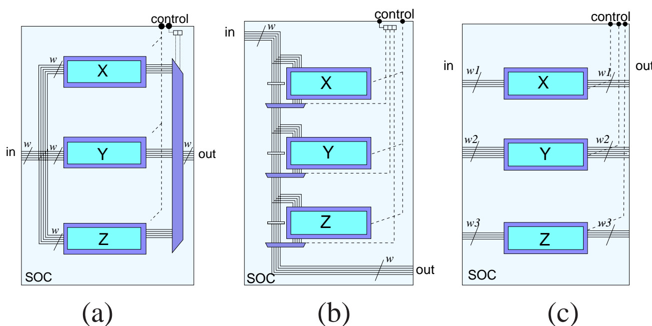


Fig. 1. Multiplexing (a), Daisychain (b), and Distribution (c) Architectures [13].

In the Multiplexing and Daisychain Architectures, all modules get access to the full available TAM width. In the Multiplexing Architecture, only one wrapper can be accessed at a time. This implies that the total test length is the sum of the individual module test lengths, but, more importantly, also that module-external testing (i.e., testing the circuitry and wiring in between the modules) is cumbersome or even impossible. This is due to the fact that only one wrapper can be accessed at a time, while for module-external testing the wrappers of two or more modules need to be accessed simultaneously. The Daisychain and Distribution Architectures do not have this restriction. In the Distribution Architecture, the total available TAM width is distributed over the modules. This allows modules to be tested concurrently, and hence the total SOC test time is the maximum of the individual module test lengths. In order to minimize the SOC test length, the width of an individual TAM should be proportional to the amount of test data that needs to be transported to and from a module connected to the TAM.

In test architecture design, we distinguish two issues: (1) the TAM type, and (2) the architecture type.

Two different *TAM types* have been proposed in the literature: the test bus and the TestRail. The *test bus*, presented by Varma and Bhatia [14], is in essence the same as what is described by the Multiplexing Architecture: modules connected to the same test bus can only be tested sequentially. The mutual exclusion for test bus access between multiple modules can for example be implemented by means of multiplexers or tri-state elements. Modules connected to a common test bus suffer from the same drawback as in the Multiplexing Architecture, viz. module-external testing is difficult or impossible.

The *TestRail*, presented by Marinissen et al. [15], is in essence the same as what is described by the Daisychain Architecture: modules connected to the same TestRail can be tested simultaneously, as well as sequentially. The TestRail can be implemented by simply concatenating the scan chains of the various modules and their wrappers. The advantage of a TestRail over a test bus is that it allows access to multiple or all wrappers simultaneously, which facilitates module-external testing.

From literature, we can distinguish at least three different *architecture types*. The first architecture type is one in which there is only one TAM, which connects to all modules. The Multiplexing and Daisychain Architectures in [13] are examples of such architectures. The Multiplexing Architecture has only one TAM of type test bus, while the Daisychain Architecture has only one TAM of type TestRail.

The second architecture type is one in which every module has its own private TAM. This architecture corresponds to the Distribution Architecture in [13]. As there is only one module per TAM, the TAM type is actually irrelevant in this type of test architecture. As each TAM needs to consist of at least one wire, this architecture requires that there are at least as many TAM wires as modules. When designing this architecture, the partitioning of the total number of available TAM wires over the various modules has a large impact on the resulting test length.

The third architecture type is the hybrid combination of types one and two. In this architecture, there are one or more TAMs, which each connects to one or more modules. This architecture is in fact a generalization, of which Multiplexing, Daisychain, and Distribution Architectures are special cases.

Test architectures based on TAM-type test bus only support *serial* test schedules; the modules connected to a common test bus are tested in an arbitrary, but sequential order [5]. Parallelism only exists in case of multiple test buses, which operate in parallel. Test architectures based on TAM-type TestRail support both *serial* and *parallel* test schedules. In a parallel test schedule, we start to test all

modules connected to a common TestRail in parallel. This continues until one of them runs out of test patterns. We then turn on the bypass for this module, while the testing of the remaining modules continues. This process is repeated until all modules on the TestRail have been completely tested. Figure 2(a) shows an example of a hybrid TestRail architecture. Figure 2(b) shows a possible corresponding serial test schedule, while Figure 2(c) shows a possible corresponding parallel test schedule.

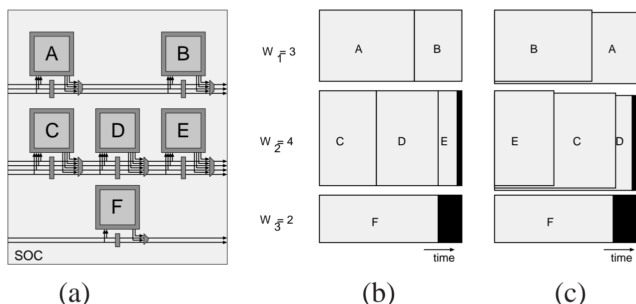


Fig. 2. Example hybrid TestRail architecture (a) and possible corresponding serial (b) and parallel (c) test schedules.

B. *Test Architecture Optimization*

Most test length minimization algorithms published so far have concentrated on hybrid test bus architectures. Chakrabarty described an architecture optimization approach that minimizes test length through Integer Linear Programming (ILP) [16] and then extended the optimization criteria with place-and-route and power constraints [17]. Edabi and Ivanov replaced ILP by a genetic algorithm [18]. In [19], Huang et al. mapped test architecture design to the well-known problem of two-dimensional bin packing and used a Best Fit Decreasing algorithm to solve it. Iyengar et al. [4] were the first to formulate the problem definition of integrated TAM/wrapper design. Despite its $\mathcal{NP}$-hard character, they solved it using ILP and exhaustive enumeration. In [20], the same authors presented efficient heuristics for it. A heuristic optimization algorithm based on rectangle packing for a test bus architecture with one single test bus that is allowed to fork and merge was presented in [21]. In [22], the same authors extended this work by including precedence, preemption, concurrency, and power constraints.

In [23], Goel and Marinissen described two heuristic algorithms for co-optimization of wrappers and TAMs for hybrid TestRail architectures. The algorithms in [23] have a limitation that the total TAM width should be greater than or equal to the number of modules inside the SOC. Therefore the approach presented in [23] is not suitable for small TAM widths.

III. Problem Definition

To design a test architecture for a given set of modules and a given number of test pins, an SOC integrator has to determine (1) the test architecture type, (2) the number of TAMs, (3) the widths of these TAMs, (4) the assignment of modules to TAMs, and (5) the wrapper design for each module. We distinguish between the problems of test architecture design for hard modules versus soft modules. Hard modules are modules for which the number and length of the module-internal scan chains are fixed and cannot be changed any more while designing the SOC-level test architecture. Examples of such modules are hard (layout) cores and encrypted cores. Soft modules are modules for which the number and length of the module-internal scan chains are not decided yet, or can still be changed while creating the SOC-level design. Examples of such modules are soft and firm cores before scan insertion.

Problem 1 [Test Architecture Design (TAD)]
Given is a set of modules M, and for each Module $m \in M$ the number of test patterns p_m, the number of functional input terminals i_m, the number of functional output terminals o_m, the number of functional bidirectional terminals b_m, the number of scan chains s_m, and for each scan chain k, the length of the scan chain in flip flops $l_{m,k}$. Furthermore is given a number $w_{\max}$ that represents the maximum number of SOC-level TAM wires that can be used. Determine a test architecture such that the overall test length is minimized and $w_{\max}$ is not exceeded. □

TAD is $\mathcal{NP}$-hard, as was shown in [4]. A variant of the above problem is the one that assumes *soft* modules, i.e., in which the module-internal scan chains are not designed yet. In that case, the number of scan chains s_m and the length of these scan chains $l_{m,k}$ are not given. Instead, for each Module m the number of scan flip flops f_m is given, and s_m and $l_{m,k}$ need to be determined such that $f_m = \sum_{k=1}^{s_m} l_{m,k}$. In practice, many SOCs actually contain a mix of hard and soft modules.

Even though the problem formulation requires data on the module-internal scan chains, this does not mean that the problem is limited to scan-testable modules only. The problem definition is equally well applicable to logic modules with full scan (where f_m equals the flip flop count of the module in question), partial scan (where f_m equals the *scan* flip flop count of the module in question), and no scan (where $f_m = 0$). The latter case is also applicable to non-logic modules, such as embedded memories, that per definition have no module-internal scan chains.

IV. TR-ARCHITECT

We have developed a tool TR-ARCHITECT [5] that designs, analyzes, and optimizes a test architecture for a given core-based SOC. Figure 3 shows the inputs and outputs of TR-ARCHITECT. TR-ARCHITECT has two inputs: a SOC data file and a list of user options. The SOC data file describes the relevant SOC parameters, such as the number of modules inside the SOC, and for each module, the number of inputs, outputs, bi-directionals, test patterns, and scan chains with their lengths. The SOC data file of TR-ARCHITECT is encoded in the so-called `*.soc` format, introduced for the *ITC'02 SOC Test Benchmarks* [24]. The user options for TR-ARCHITECT include (but are not limited to): (1) total number of SOC test pins, (2) type of modules (hard/soft), (3) test schedule type (serial/parallel), (4) TAM type (test bus/TestRail), (5) architecture type (daisychain/hybrid/distribution), and (6) test cost.

Fig. 3. Input/output flow of TR-ARCHITECT.

TR-ARCHITECT supports the design of three types of architectures: (1) the Daisychain Architecture [13], (2) the Distribution Architecture [13], and (3) the Hybrid Architecture [5, 7]. In case of a Daisychain Architecture, all modules in the SOC are connected to a single TAM with full bandwidth. In a Distribution Architecture, all modules have disjunct TAMs; TR-ARCHITECT optimally distributes the total TAM width among all modules using the SCDP algorithm [13]. For the Hybrid Architecture, TR-ARCHITECT uses a four-step algorithm described in [5, 7] to determine the number of TAMs, their widths, and module-to-TAM assignments. To design a wrapper around a module for a given TAM width, TR-ARCHITECT uses the COMBINE algorithm presented in [3].

Based on the test parameters selected by the user, TR-ARCHITECT yields (a) a test architecture and (b) an optimal test schedule for this architecture, such that the total test cost is minimal. The basic version of TR-ARCHITECT [5] minimizes the SOC test length only. Later, we have step-wise extended this basic version to include the following practical constraints related to real-life SOC designs.

A. Layout Constraints

In an era of designing multi-million-transistor SOCs with deep-submicron technology, the wiring of TAMs is another important cost. Nowadays, many SOCs implement quite wide, dedicated TAMs. Especially for such SOCs, it pays off to take into account the minimization of the TAM *wire length* while designing the test architecture. Short TAM wires reduce the required area cost, performance impact, power dissipation, and cross-coupling between the TAM wires. To optimize the TAM wire length, a test architecture design procedure should take into account the layout positions of all modules in the SOC. Next to that, the procedure should try to assign the neighboring modules to the same TAM as much as possible. If the layout positions of modules in an SOC are not considered while designing the test architecture for the SOC, modules positioned at different corners in the SOC-layout may get assigned to the same TAM. To connect these modules through TAM wires, TAM wires often go across the entire layout (perhaps even multiple times). Therefore, in order to optimize the TAM wire length, a test architecture design procedure should take into account the layout positions of all modules in the SOC.

To minimize the total wire length required for routing the TAM wires in the designed test architecture, layout-constraints have been included in TR-ARCHITECT [6]. To calculate the total wire length for a test architecture, we use a simple yet effective wire-length cost model as described in [6]. In our wire length model, we assume that every module is a single point and distances between modules and from the chip boundaries are measured from the corresponding points. Furthermore, we assume that the pins to which TAM wires will be connected are not predetermined and can be changed.

Minimization of wire length requires two tasks to be solved by TR-ARCHITECT. The first task is to partition the modules among different TAMs. The second task is to order the modules within each TAM, as the wire length of a TAM depends on the ordering of the modules connected to the TAM. As the minimization of test length and wire length can be conflicting, a weight-based cost function has been used to partition the modules among TAMs. Now, TR-ARCHITECT tries to assign neighboring modules as much as possible to the same TAM. The second task, i.e., the problem of determining an optimal ordering of modules with respect to the wire length of the TAM is equivalent to the well-known Traveling Salesman Problem

(TSP). Therefore, we use a simple TSP heuristic algorithm to solve it. Experimental results for several benchmark SOCs [24] show that the layout-driven TR-ARCHITECT can save upto 85% in TAM wire length at an expense of less than 5% in test length.

B. Test Control Constraints

Based on the nature of test-control signals, test control can be classified into two categories: (1) pseudo-static test control, and (2) dynamic test control. Pseudo-static test-control signals are often provided by means of a shift-register (WIR), where as dynamic test-control signals require dedicated test pins. Before starting a test of a module, the wrapper of the module needs to be configured in the corresponding test mode. Setting a particular test mode in the wrapper of a module requires a few pseudo-static test-control signals, while parallel test execution of modules connected to different TAMs requires at-least one *scan-enable* (dynamic signal) per TAM. Setting a particular test mode in the wrapper of a module requires extra time, while a scan-enable signal requires an extra pin. As the total number of chip pins available for test is limited, a large number of test-control pins will result in less TAM bandwidth available for test-data transportation. Therefore, test architecture design should take the test control into account.

In [8], we have extended TR-ARCHITECT to take into account the test control required for the test architecture. The term test control refers to controlling the mode of operation of all modules and the execution of their tests. To account for the time required for test mode setting, two test strategies are presented: (1) one WIR chain per SOC, and (2) one WIR chain per TAM. Each of these strategies provides a trade-off between the time to select a WIR chain and programming the WIR chain. To minimize the extra pins required for scan enable signals, TR-ARCHITECT has been modified in such a way that it already takes into account one pin per TAM for the scan enable signal for that TAM.

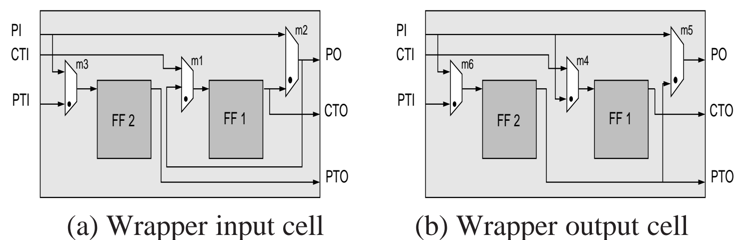
C. Support for Hierarchical SOCs

Modern SOC designs are not limited to only one level of hierarchy (SOC and modules), instead they consist of multiple levels of design hierarchy. For example, [12, 25] describe SOCs for digital video, for which the design is partitioned into design units called *chiplets*, which in turn consist of modules. Based on the hierarchical relation, a hierarchical module is also called a *parent module*, while the modules which are at one-level below and embedded in this module, are called *child modules*. While designing test architectures for SOCs with hierarchical modules, wrappers for hierarchical modules have to be designed such that test access is available to both the parent and child modules. Testing of a parent module requires access to its own elements such as scan chains and wrapper cells as well as to the wrapper cells of its child modules.

For simplicity, TR-ARCHITECT so far have assumed only one level of hierarchy (SOC and modules), i.e. even if there is a hierarchy among the modules, all modules in an SOC are treated at the same level of hierarchy. Due to this assumption, TR-ARCHITECT designs wrappers for both hierarchical and flat modules in the same fashion. For parent module testing, TR-ARCHITECT does not consider the access required to the elements in their child modules. Furthermore, optimal test schedules proposed by TR-ARCHITECT allow parallel testing of parent and child modules, which is *not* supported by the existing wrapper architectures [14, 15, 26].

There are two ways in which hierarchical constraints can be handled while designing a test architecture: (1) modify the test hardware such that it gives full flexibility to the algorithm, and (2) do not modify the hardware but modify the test architecture design algorithm. Both solutions have their own advantages and disadvantages.

In [9], we have presented an approach to design test architectures for SOCs with hierarchical modules using existing TR-ARCHITECT. Here, we did not make any major modification to TR-ARCHITECT, but we modified the modules wrappers in such a way that all hierarchical constraints are satisfied. To allow testing of both the parent and child modules in parallel, an improved wrapper architecture is presented. In the improved wrapper, a new type of wrapper cell is used in the child module wrapper. Unlike the conventional wrapper cell which is connected to only one TAM, the new wrapper cell connects to both the parent TAM and the child TAM. Furthermore, instead of one flip flop, the new wrapper cell contains two flip flops. Figure 4 shows an example implementation of the proposed wrapper cell.



(a) Wrapper input cell (b) Wrapper output cell

Fig. 4. Example implementation of the proposed wrapper cells.

Furthermore, test length calculations in TR-ARCHITECT has been slightly modified to take into account the time required to access the elements in all child modules while testing the corresponding parent module. By using the

proposed wrapper architecture, optimal test schedules obtained from the existing TR-ARCHITECT that consider SOCs with flat modules can be implemented directly for the same SOCs with hierarchical modules. Hence, optimal test lengths can be obtained for SOCs with hierarchical modules. The only drawback of the proposed approach is the extra silicon area due to large wrapper cells.

To minimize the silicon area, we are looking at designing optimal wrappers for hierarchical modules [11]. Here, we focus on designing a wrapper for a hierarchical module such that the hierarchical constraints are satisfied and also the wrapper does not require extra silicon area. This research is still going on and we expect to report new results in the coming future.

D. User Constraints

TR-ARCHITECT designs an SOC test architecture fully autonomously. This can lead to situations where test architectures proposed by TR-ARCHITECT might not be acceptable to SOC designers due to some design constraints which were either not modeled or very hard to model directly in the optimization procedure. For example, a wrapped third-party IP module requires a specific TAM width; analog and digital modules or modules running at different clock frequencies need to be assigned to separate TAMs; some test architecture design constraints inherited from a previous SOC design. We have learned in practice that SOC designers and test engineers do not always want to leave their entire SOC-level test architecture design to an optimization tool. They often want to influence the number of TAMs, the width of the TAMs, the assignment of modules to TAMs, and/or the modules ordering within a TAM. As the SOC designers have the ultimate decision power over *their* SOC, a test architecture optimization tool can only become successful in the SOC designer community if it is able to satisfy their constraints.

To include a wide range of user constraints which might be application- or design-specific and hence hard to generalize into general cost functions, we have developed a novel Test Architecture Specification (TAS) language [10]. In a TAS file, a user can fully or partially specify test architecture parameters. Figure 5 shows an example TAS file for SOC p22810 [24]. Please note that the line numbers in the example are not part of the TAS language, but are added here for explanation purposes only.

The first line in the TAS example identifies the SOC, viz. SOC p22810. Line 3 specifies the total number of TAMs in the architecture. In this example, the minimum number of TAMs is specified as 3, while the maximum number of TAMs is specified as 5. The next three lines (Lines 4–6) specify three TAMs, corresponding to the minimum number of three TAMs specified in Line 3. Line 4 specifies TAM $ru1$. Its minimum specified width is 4; no maximum width is specified. The maximum number of modules that are allowed to connect to this TAM is set to 3. Module 18 is specified to be the first module in this TAM. During test architecture design, the used architecture design algorithm has to add one or more modules (as specified by the Kleene plus (+) operator) after module 18, as long as TAM $ru1$ in total does not have more than three modules (as specified by the `MaxModules` constraint). TAM $ru2$ is specified in Line 5. Its width is specified to be between 1 and 9. TAM $ru2$ can contain at most four modules, and modules 4 and 5 are amongst those. The specified order of the TAM states that module 5 comes in front of module 4, while additional modules (if any) can be placed anywhere in front, in between, and after those two modules. Line 6 gives a similar specification for TAM $ru3$.

Example 1 [p22810.tas]

```
1  SocName p22810
2  // TAM Specification
3  TotalTAMs 3-5
4  TAM ru1 Width 4-  MaxModules 3 FixModules:18     Order:18-+
5  TAM ru2 Width 1-9 MaxModules 4 FixModules:4,5    Order:*-5-*-4-*
6  TAM ru3 Width 5-  MaxModules 5 FixModules:6,7,9  Order:*-9-7-6-*
7  // Module Specification
8  Module 2 FlexTAMs :  ru2,ru3,rx
```

Fig. 5. Example TAS file for SOC p22810.

In total, only six modules are pre-assigned to the three user-defined TAMs $ru1$, $ru2$, and $ru3$. From [24], it is known that SOC p22810 contains 28 modules (in the benchmark set, modules are referred to as modules). Therefore, the assignment of the remaining 22 modules is left to the test architecture design algorithm, within the constraints specified in Lines 8–9. Line 8 states that module 2 is only allowed to be assigned to either TAM $ru2$ or TAM $ru3$ or TAM rx (the later represents a TAM created by the test architecture design algorithm itself).

TR-ARCHITECT has been extended to take into account the user constraints specified in a input TAS file. Figure 6 shows the input and output files used for the extended TR-ARCHITECT. Apart from the two regular inputs, there is a third new (optional) input TAS file, which contains the user constraints in the TAS language. TR-ARCHITECT yields (a) a test architecture which is optimized within the bounds of the user constraints, (b) an optimal test schedule for this architecture, and (c), new, the corresponding TAS file. If for some reason the user does not like the proposed architecture, he/she can take the output TAS file, modify it, and feed it back again for a new TR-ARCHITECT run.

All parameters specified by the user are considered as constraints, while everything which is not specified, is left

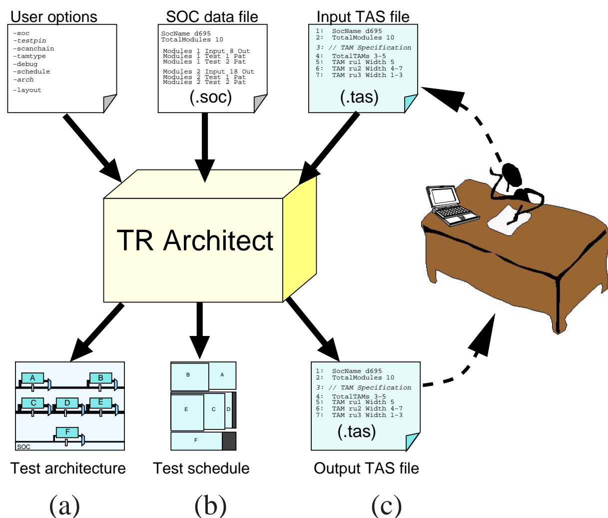
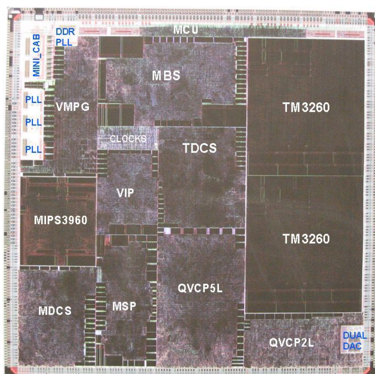


Fig. 6. User-constrained TR-ARCHITECT.

for the tool to optimize for minimal test length, TAM wire length, etc. This extension yields a whole spectrum of use scenarios for TR-ARCHITECT. The spectrum ranges from an empty user specification (in which the tool fully optimizes the resulting test architecture) to a full user specification (in which the tool only computes the corresponding test schedule and associated costs) and everything in between.

V. INDUSTRIAL CASE STUDY

First practical use of TR-ARCHITECT has been reported in [12] for the Philips PNX8550 complex SOC. The PNX8550 is based on the Nexperia™ Home Platform. It contains about ten million gates. In total, the PNX8550 contains 62 logic IP blocks, out of which five are hard cores while the rest are soft cores. The five hard cores includes one MIPS RISC CPU and two VLIW TriMedia CPUs. Figure 7 shows its layout and characteristics. The PNX8550 is a chiplet based design. A chiplet is a design entity consisting of one or more cores.



0.13μm CMOS process
6 metal layers
1.2 V supply voltage
PBGA564 package
100 mm^2 die size
10M logic gates
40M logic transistors
338,859 flip-flops
62 logic cores
212 memory cores
94 clock domains
140 TestRail wires
full scan, BIST, and functional testability

Fig. 7. PNX8550 chip layout and characteristics.

We consider three cases [12]. Case 1 is the original test architecture implemented on the PNX8550, in which the TAM width assignment to the chiplets was done manually. All chiplets have a Distribution Architecture, except for two chiplets, which have a Hybrid Architecture. The architectures for all chiplets were designed by TR-ARCHITECT without modifying the original number and lengths of the core-internal scan chains.

In Case 2, the distribution of 140 TAM wires over the 13 chiplets was optimized by TR-ARCHITECT. Also in these cases, we did not modify the original number and lengths of the core-internal scan chains. In this case, all chiplets were allowed to have a Hybrid Architecture (which in some cases might end up becoming either a Distribution or Daisychain Architecture).

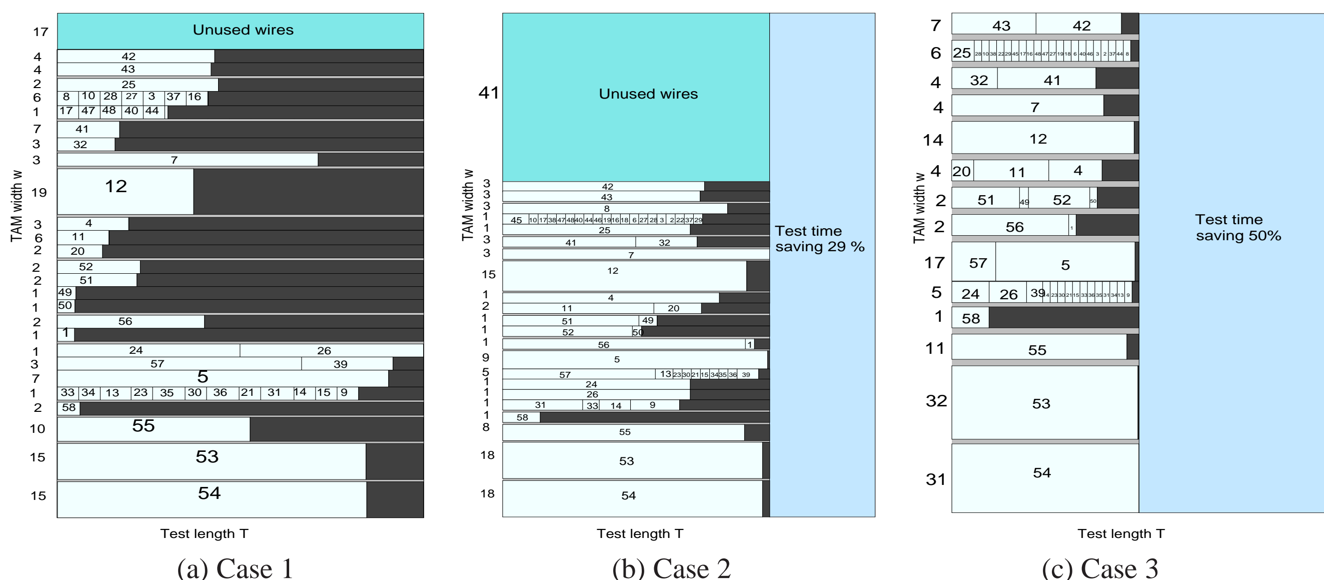
Case 3 is equal to Case 2, apart from the fact that we allowed TR-ARCHITECT to modify and optimize the number and lengths of the core-internal scan chains of all cores, except the Philips-external hard cores TriMedia (two instances) and MIPS.

Figure 8 shows the corresponding test schedules for the three considered cases. In the figure, the horizontal axes displays the test length, while the vertical axes shows the TAM width; the latter is not to scale. The light numbered boxes depict the tests of the cores; the number in the box is the core ID. The horizontal light-grey lines separate the test schedules of the individual chiplets. TR-ARCHITECT was only available halfway the design trajectory of PNX8550 SOC. The target ATE for the PNX8550 was an Agilent 93000-P600 test system with 28 M vector memory per channel. The total test length for the test schedule shown in Figure 8(a) is 3,506,193 clock cycles. Here, we only consider one clock domain per chiplet. After taking care of the testing with multiple clock domains, this number translated in a total test data volume that fitted onto the target ATE. Figure 8(b) and (c) show that, if TR-ARCHITECT would have been available from the project start onwards, further test length reductions up to 50% would have been possible.

VI. CONCLUSION

For modular test development, test architectures consisting of TAMs and wrappers are required. In this paper, we described a tool TR-ARCHITECT that designs and optimizes SOC test architectures with respect to various test costs. TR-ARCHITECT optimizes wrapper and TAM design in conjunction. TR-ARCHITECT handles both SOCs with hard and soft cores, works for test bus and TestRail TAM types, and supports both serial and parallel schedules.

The basic version of TR-ARCHITECT that optimizes SOC test architectures with respect to the overall SOC test



(a) Case 1 (b) Case 2 (c) Case 3

Fig. 8. Test schedules of the various test architecture cases of PNX8550.

length has been step-wise extended to include the following practical constraints that related to real-life SOC designs. These constraints are layout, test control, design hierarchy, and user control. First practical use of TR-ARCHITECT has been reported for the Philips PNX8550 SOC. TR-ARCHITECT was only available halfway the design trajectory of PNX8550 SOC. Despite of that it helped to optimize the test architecture further within the given constraints and we successfully managed to fit the test date onto the target ATE. Experimental results show that, if TR-ARCHITECT would have been available from the project start onwards, further test length reductions up to 50% would have been possible.

VII. ACKNOWLEDGEMENTS

We thank M.Sc. student Ludovic Krundel of ISIM, University of Montpellier II, France, and his supervisors Marie-Lise Flottes and Bruno Rouzeyre of LIRMM, France for their contribution to the work related to inclusion of user constraints in TR-ARCHITECT. We thank Ph.D. student Anuja Sehgal and her supervisor Krishnendu Chakrabarty of Duke University, Durham, NC, USA for their contribution to the work related to inclusion of hierarchical constraints in TR-ARCHITECT. We thank Kuoshu Chiu, Toan Nguyen, and Steven Oostdijk of Philips Semiconductors, San Jose, CA, USA for their contributions regarding the first use of TR-ARCHITECT on the Philips PNX8550 SOC. Furthermore, we thank our colleagues Bart Vermeulen and Harald Vranken for their useful comments on an earlier draft version of this paper.

REFERENCES

[1] Rajesh K. Gupta and Yervant Zorian. Introducing Core-Based System Design. *IEEE Design & Test of Computers*, 14(4):15–25, December 1997.

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

[3] Erik Jan Marinissen, Sandeep Kumar Goel, and Maurice Lousberg. Wrapper Design for Embedded Core Test. In *Proceedings IEEE International Test Conference (ITC)*, pages 911–920, Atlantic City, NJ, October 2000.

[4] Vikram Iyengar, Krishnendu Chakrabarty, and Erik Jan Marinissen. Co-Optimization of Test Wrapper and Test Access Architecture for Embedded Cores. *Journal of Electronic Testing: Theory and Applications*, 18(2):213–230, April 2002.

[5] 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.

[6] Sandeep Kumar Goel and Erik Jan Marinissen. Layout-Driven SOC Test Architecture Design for Test Time and Wire Length Minimization. In *Proceedings Design, Automation, and Test in Europe (DATE)*, pages 738–743, Munich, Germany, March 2003.

[7] Sandeep Kumar Goel and Erik Jan Marinissen. SOC Test Architecture Design for Efficient Utilization of Test Bandwidth. *ACM Transactions on Design Automation of Electronic Systems*, 8(4):399–429, October 2003.

[8] Sandeep Kumar Goel and Erik Jan Marinissen. Control-Aware Test Architecture Design for Modular SOC Testing. In *Proceedings IEEE European Test Workshop (ETW)*, pages 57–62, Maastricht, The Netherlands, May 2003.

[9] Sandeep Kumar Goel. A Novel Wrapper Cell Design for Efficient Testing of Hierarchical Cores in System Chips. In *Proceedings IEEE European Test Symposium (ETS)*, pages 147–152, Ajaccio, Corsica, May 2004.

[10] Ludovic Krundel, Sandeep Kumar Goel, Erik Jan Marinissen, Marie-Lise Flottes, and Bruno Rouzeyre. User-Constrained Test Architecture Design for Modular SOC Testing. In *Proceedings IEEE European Test Symposium (ETS)*, pages 80–86, Ajaccio, Corsica, May 2004.

[11] Anuja Sehgal, Sandeep Kumar Goel, Erik Jan Marinissen, and Krishnendu Chakrabarty. P1500-Compliant Test Wrapper Design for Hierarchical Cores. In *Proceedings IEEE International Test Conference (ITC)*, Charlotte, NC, 2004. (To appear).

[12] Sandeep Kumar Goel, Kuoshu Chiu, Erik Jan Marinissen, Toan Nguyen, and Steven Oostdijk. Test Infrastructure Design for the Nexperia^{TM} Home Platform PNX8550 System Chip. In *Proceedings Design, Automation, and Test in Europe (DATE)*, pages 108–113, February 2004.

[13] Joep Aerts and Erik Jan Marinissen. Scan Chain Design for Test Time Reduction in Core-Based ICs. In *Proceedings IEEE International Test Conference (ITC)*, pages 448–457, Washington, DC, October 1998.

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

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

[16] Krishnendu Chakrabarty. Design of System-on-a-Chip Test Access Architectures Using Integer Linear Programming. In *Proceedings IEEE VLSI Test Symposium (VTS)*, pages 127–134, Montreal, Canada, April 2000.

[17] Krishnendu Chakrabarty. Design of System-on-a-Chip Test Access Architectures Under Place-and-Route and Power Constraints. In *Proceedings ACM/IEEE Design Automation Conference (DAC)*, pages 432–437, Los Angeles, CA, June 2000.

[18] Zahra Sadat Ebadi and Andre Ivanov. Design of an Optimal Test Access Architecture Using a Genetic Algorithm. In *Proceedings IEEE Asian Test Symposium (ATS)*, pages 205–210, Kyoto, Japan, November 2001.

[19] Yu Huang et al. Resource Allocation and Test Scheduling for Concurrent Test of Core-Based SOC Design. In *Proceedings IEEE Asian Test Symposium (ATS)*, pages 265–270, Kyoto, Japan, November 2001.

[20] Vikram Iyengar, Krishnendu Chakrabarty, and Erik Jan Marinissen. Efficient Wrapper/TAM Co-Optimization for Large SOCs. In *Proceedings Design, Automation, and Test in Europe (DATE)*, pages 491–498, Paris, France, March 2002.

[21] Vikram Iyengar, Krishnendu Chakrabarty, and Erik Jan Marinissen. On Using Rectangle Packing for SOC Wrapper/TAM Co-Optimization. In *Proceedings IEEE VLSI Test Symposium (VTS)*, pages 253–258, Monterey, CA, April 2002.

[22] Vikram Iyengar, Krishnendu Chakrabarty, and Erik Jan Marinissen. Integrated Wrapper/TAM Co-Optimization, Constraint-Driven Test Scheduling, and Tester Data Volume Reduction for SOCs. In *Proceedings ACM/IEEE Design Automation Conference (DAC)*, pages 685–690, New Orleans, LO, June 2002.

[23] Sandeep Kumar Goel and Erik Jan Marinissen. Cluster-Based Test Architecture Design for System-on-Chip. In *Proceedings IEEE VLSI Test Symposium (VTS)*, pages 259–264, Monterey, CA, April 2002.

[24] Erik Jan Marinissen, Vikram Iyengar, and Krishnendu Chakrabarty. A Set of Benchmarks for Modular Testing of SOCs. In *Proceedings IEEE International Test Conference (ITC)*, pages 519–528, Baltimore, MD, October 2002. (See http://www.extra.research.philips.com/itc02socbenchm/).

[25] Santanu Dutta, Rune Jensen, and Alf Rieckmann. VIPER: A Multiprocessor SOC for Advanced Set-Top Box and Digital TV Systems. In *IEEE Design & Test of Computers*, pages 21–31, 2001.

[26] Francisco DaSilva, Yervant Zorian, Lee Whetsel, Karim Arabi, and Rohit Kapur. Overview of the IEEE P1500 Standard. In *Proceedings IEEE International Test Conference (ITC)*, pages 988–997, Charlotte, NC, September 2003.