Test Access Mechanism Optimization, Test Scheduling, and Tester Data Volume Reduction for System-on-Chip

Vikram Iyengar, *Student Member*, *IEEE*, Krishnendu Chakrabarty, *Senior Member*, *IEEE*, and Erik Jan Marinissen, *Senior Member*, *IEEE*

Abstract—We describe an integrated framework for system-on-chip (SOC) test automation. Our framework is based on a new test access mechanism (TAM) architecture consisting of flexible-width test buses that can fork and merge between cores. Test wrapper and TAM cooptimization for this architecture is performed by representing core tests using rectangles and by employing a novel rectangle packing algorithm for test scheduling. Test scheduling is tightly integrated with TAM optimization and it incorporates precedence and power constraints in the test schedule, while allowing the SOC integrator to designate a group of tests as preemptable. Test preemption helps avoid hardware and power consumption conflicts, thereby leading to a more efficient test schedule. Finally, we study the relationship between TAM width and tester data volume to identify an effective TAM width for the SOC. We present experimental results on our test automation framework for four benchmark SOCs.

Index Terms—Core-based systems, rectangle packing, system-on-a-chip, test access mechanism, test scheduling, testing time, test wrapper.

1 Introduction

Modular testing of embedded cores in a system-on-chip (SOC) is now recognized as an effective method to tackle the SOC testing poblem. Modular test refers to the process of isolating embedded cores and groups of cores and gaining test access to these separate SOC partitions for test data delivery. The primary motivation for modular test is to use a "divide-and-conquer" approach and to promote the reuse of precomputed test sets for embedded cores. In addition, for nonlogic cores, such as memories or intellectual property (IP) cores, for which vendor-computed tests are mandatory, isolation and modular test is often the only option. To facilitate modular test, an embedded core must be isolated from surrounding logic and test access must be provided from the I/O pins of the SOC [26]. Test wrappers are used to isolate the core for modular test application, while test access mechanisms (TAMs) transport test patterns and test responses between SOCs pins and core I/Os [20]. In addition, core tests must be scheduled such that precedence and power constraints are met and conflicts in TAM usage are avoided.

To reduce cost and ensure quality, testing must make effective use of SOC test resources [4]. The rapidly increasing size of SOCs has spurred an enormous growth in test resource usage, leading to complex test hardware, long test application times, and large test data. An integrated framework for SOC test automation that increases the utilization of test resources is therefore necessary to increase production capacities and reduce test cost. This paper describes the design of such a framework that integrates the following three design processes into the test automation flow.

1. **Wrapper/TAM cooptimization.** Our framework maximizes the effectiveness of on-chip test structures. Test wrapper design and TAM optimization are of critical importance during system integration since they directly impact hardware costs, testing time, and tester data volume. Our TAM architecture is based on flexible-width test buses. The new approach proposed for TAM optimization uses a generalized version of rectangle packing [7].
2. **Constraint-driven preemptive test scheduling.** The test automation framework integrates an efficient test scheduling algorithm with the TAM design process. The objective of the scheduling algorithm is to minimize the testing time, while addressing the following issues: a) resource conflicts between cores arising from the use of shared TAMs and on-chip BIST engines, b) precedence constraints among tests, and c) power consumption constraints. Testing time is further decreased through the selective use of test preemption.
3. **Tester data volume reduction.** The third component in our framework addresses the issue of identifying an SOC TAM width that reduces testing time as well as tester data volume. Test data for large SOCs now require several Gigabits of tester memory, which is a

- *V. Iyengar is with IBM Microelectronics, 1000 River Rd., Bldg. 863 B, Essex Junction, VT 05452. E-mail: vikrami@us.ibm.com.*
- *K. Chakrabarty is with the Electrical and Computer Engineering Department, Duke University, Durham, NC 27708. E-mail: krish@ee.duke.edu.*
- *E.J. Marinissen is with Philips Research Laboratories, Prof. Holstlaan 4, M/S WAY-41, 5656 AA Eindhoven, The Netherlands. E-mail: Erik.Jan.Marinissen@philips.com.*

Manuscript received 7 Aug. 2002; revised 26 Feb. 2003; accepted 17 Apr. 2003.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number 117086.

significant contributor to test cost [2], [4]. The impact of TAM design and test schedules on tester memory requirements has not been directly studied before. In particular, we show that TAM widths that minimize testing time do not always lead to minimum tester data volume. We investigate the correlation of TAM width to the tester data volume,and determine TAM widths that minimize a cost function involving both the testing time and the amount of tester data. This allows the system integrator to trade off testing time with data volume.

The remainder of this paper is organized as follows: In Section 2, we discuss related prior work. In Section 3, we describe the new flexible-width TAM architecture. In Section 4, we define the integrated TAM optimization/test scheduling problem and formulate it as a version of generalized rectangle packing. In Section 5, we present the new approach to constraint-driven test scheduling. Our framework provides the system integrator with the ability to specify a partial precedence ordering among the core tests as well as ensure that test parallelism does not violate constraints on power consumption. Furthermore, the integrator is provided with the means to identify certain tests for preemption to minimize testing time. In Section 6, we describe the impact of TAM width on tester data volume. Experimental results for four benchmark SOCs are presented in Section 7. Section 8 concludes the paper.

2 Related Prior Work

Here, we review prior work in TAM design, test scheduling, and tester data volume reduction.

Prior work in TAM architecture design includes a dedicated test bus [24], partial isolation rings [23], reuse of the system bus [10], a scalable architecture called TestRail [18], and a P1500-compatible TAM known as CAS-BUS [3]. The TestRail and test bus architectures appear to be the most amenable to optimization because a broad range of algorithmic techniques [9], [11], [13] can be employed to optimize the TestRail and test bus widths to minimize the SOC testing time. However, most proposed TAM optimization methods have studied wrapper design and TAM optimization as independent problems [4], [19]. These methods have either not addressed wrapper design [1], [11], [17] or not explicitly addressed the issue of sizing TAMs to minimize SOC testing time [22].

The first integrated wrapper/TAM cooptimization methods to minimize SOC testing time were proposed in [13], [14], [16]. A drawback of the approach in [13] is that the problem formulation is intrinsically intractable; the computation time therefore increases exponentially with the number of TAMs. In [14], an efficient heuristic was proposed to address the wrapper/TAM design problems in [13]. The heuristic method was able to prune the solution-space and achieve significant reductions in CPU time over the method of [13]. Furthermore, in [14], it was possible to design test architectures with a larger number of TAMs and, therefore, reductions in testing time were also achieved in a few cases. In [16], a graph formulation of the TAM design and test scheduling problem was presented. An algorithm based on bipartite graph matching was used to optimize the test bus architecture.

However, the approaches in [13], [14], [16] are limited to "fixed-width" test bus architectures. In fixed-width TAMs, the total TAM width is explicitly partitioned among a finite number of TAMs and each core is assigned to a TAM. Therefore, a large number of cores of varying TAM width requirements are often assigned to a small number of TAMs. Since TAM widths cannot be explicitly tailored to each core's requirement, wasteful TAM wires are often assigned to certain cores. For example, if a core is connected to a TAM of width w, the same testing time may actually be obtained using only $w' < w$ wires (due to the "staircase" nature of the core's testing time variation with TAM width; see Section 3). This leads to unneccessary hardware overhead. This also leads to idle bits stored on the tester for the extra $w - w'$ wires, hence increasing tester data volume as well. In this paper, we design "flexible-width" test buses for cores. In a flexible-width TAM architecture, the TAM width supplied to each core is based explicitly on the core's TAM width needs; each core can be assigned a unique number of TAM wires corresponding to its testing time versus TAM width function. This leads to a more effective TAM design.

Several techniques for SOC test scheduling have been proposed in the literature [4], [6], [15], [17], [20], [25]. Methods to incorporate precedence and power constraints in a preemptive test schedule were presented in [12]. While these methods are useful to obtain test schedules, they assume that a predesigned TAM for the SOC is provided. Here, we address the design of an integrated framework for SOC test automation where TAM optimization and test scheduling are performed in *conjunction*.

Tester data volume reduction methods are either based on built-in self test (BIST) [5] or test data compression [4]. BIST can be used effectively to reduce the volume of test data stored in tester memory by generating test patterns directly on-chip. However, the cores for which the system integrator can insert BIST are mostly limited to certain types of memories. For logic cores and hard IP, if BIST is desired, it must be inserted by core vendors. Moreover, BIST can be expensive in terms of hardware costs and may not be feasible for cores that are not BIST-ready. While test set compression has been shown to be useful for test data volume reduction, there has not been a concerted effort to examine its use coupled with TAM design and test scheduling in the form of an integrated framework for test automation. TAM widths optimized for test data volume can reduce tester load time and facilitate multisite testing.

In this paper, we present a new approach to integrated wrapper/TAM cooptimization and test scheduling based on a generalized version of rectangle packing, also referred to as two-dimensional packing [7]. Rectangle representation of core tests has previously been studied in [6], [17] and a method based on rectangle packing was proposed to schedule tests for SOCs in [11]. We use the wrapper design method presented in [13] to design a wrapper for each core based on the knowledge of how the core's testing time varies with TAM width. We design flexible-width TAMs by assigning an appropriate number of TAM wires to each core in the test schedule. Therefore, the granularity of TAM

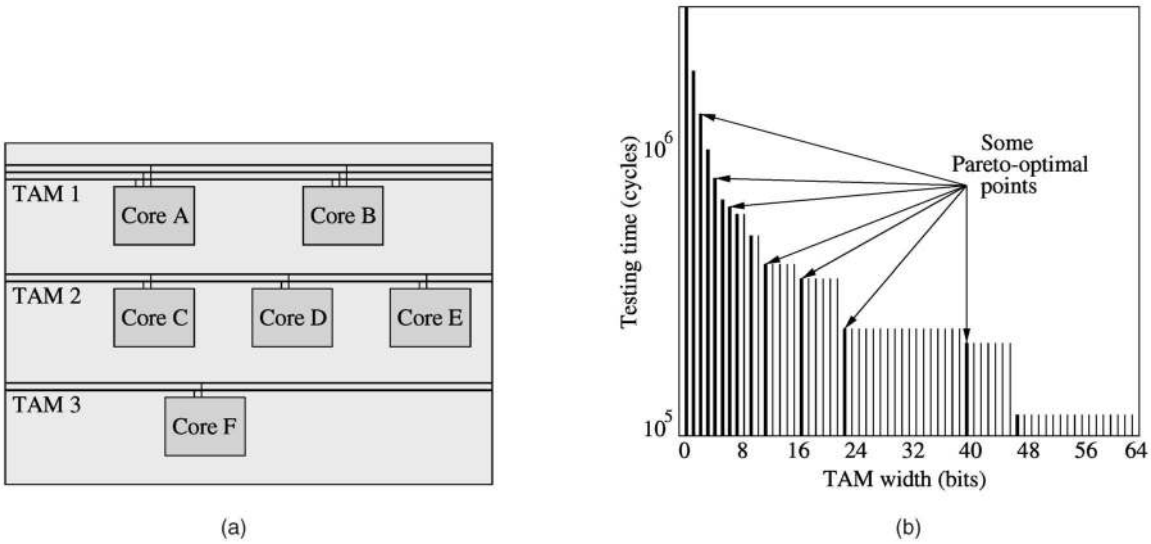

Fig. 1. Illustration of (a) TAM design for fixed-width architecture and (b) relationship between testing time and TAM width for Core 6 in benchmark SOC p93791 [21].

assignment to cores is *per wire* and not *per test bus* as in the fixed-width case. This fine-grained TAM wire assignment leads to more efficient TAM architectures. Precedence constraints among tests can be embedded into the schedule by the system integrator and the maximum power consumption limit can be specified. Moreover, the system integrator can identify certain tests as preemptable, as well as specify limits on the number of preemptions allowed for each test. Finally, the relationship between TAM width and tester data volume is investigated to identify an effective choice of total TAM width for the SOC. Note that other flexible-width TAM architectures can also be designed, such as a flexible-width TestRail architecture. In this work, we consider the flexible-width test bus architecture, based on our prior work on using test buses for TAM design.

3 Flexible-Width Test Bus Architecture

The test bus model for TAMs was first proposed in [24]. In this model, a TAM consists of a set of wires that can be grouped into a test bus and connected to cores. The TAMs operate independently of each other; however, the cores on a single TAM are tested sequentially. This can be implemented either by 1) multiplexing all the cores assigned to a TAM or 2) by testing one of the cores on the TAM, while the other cores on the TAM are bypassed.

In [13], [14], TAMs were modeled as fixed-width test buses. In a fixed-width test bus architecture, the total TAM width is partitioned among several test buses. For example, a total TAM width of 7 can be partitioned among three test buses as follows: $3 + 2 + 2$. Each core in the SOC is assigned to exactly one of these fixed-width test buses. TAM assignment is therefore on a test bus basis. This is illustrated in Fig. 1a for an example SOC containing six cores.

There are three main drawbacks of fixed-width architectures that limit the choice of available TAM widths for cores. First, each core does not receive a TAM width explicitly tailored to its own requirements. The TAM width assigned to each core is based on the overall requirement of the group of cores assigned to the test bus. Second, the maximum TAM width available to any core equals the width of the widest test bus in the SOC. Therefore, to increase TAM width per core, the number of TAMs is reduced and a large number of cores are assigned to only a few TAMs. This further reduces the choice of TAM widths available to any individual core. Third, the computation time for the exact method in [13] grows exponentially with the number of TAMs, therefore solutions can be obtained for only a small number of TAMs, again limiting the number of available TAM width choices for cores.

We next introduce the concept of Pareto-optimal TAM widths to show how flexible-width TAM architectures can be designed in which TAM widths are explicitly tailored to the requirements of each core, thereby minimizing hardware overhead, testing time, and test data volume.

Pareto-optimal points. We achieve a significant improvement in TAM wire utilization over the method in [13] by noting that only a few TAM widths between 1 and W, where W is the total SOC TAM width, are efficient if assigned to cores. It was shown in [13] that, for a given core, the testing time varies with TAM width as a "staircase" function. From Fig. 1b, we see that the testing time decreases only at Pareto-optimal points, which are formally defined as follows: A solution to the wrapper design problem for Core i can be expressed as a 2-tuple $(w_j, T_i(w_j))$, where w_j is the TAM width supplied to the wrapper and $T_i(w_j)$ is the testing time of Core i with the given wrapper. A solution $(w_j, T_i(w_j))$ is *Pareto-optimal* if and only if there does not exist a solution $(w_k, T_i(w_k))$ such that $w_k \leq w_j$ and $T_i(w_k) \leq T_i(w_j)$, where at least one of the inequalities is strict [8]. Intuitively, the steps at which the testing time decreases (as TAM width is increased) are the Pareto-optimal points and only Pareto-optimal TAM width values need to be considered. For example, in Fig. 1b, a TAM width of 46 results in a testing time of 115,850 cycles, while all TAM widths from 47 up to 64 result in the same testing time of 114,317 cycles. Hence, 47 is a Pareto-optimal TAM width and widths between 48 and 64 can be ignored.

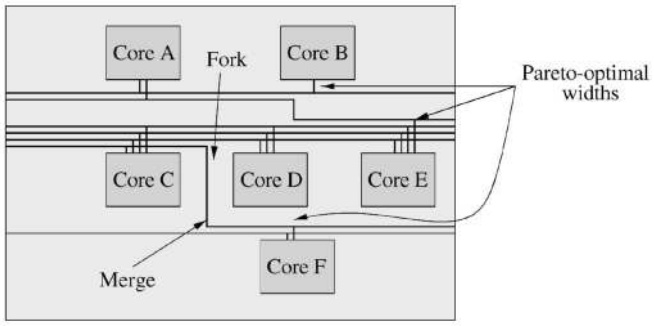

Fig. 2. Example TAM design for flexible-width architecture.

Since the number of available TAM widths for cores is limited in fixed-width architectures, cores are often assigned to non-Pareto-optimal TAM widths. These wasteful assignments increase hardware overhead unneccesarily. Furthermore, the tester is required to store don't-care (idle) bits that must be transported to the wasteful TAM wires during test, thereby increasing test data volume and testing time.

Flexible-width test bus architectures allow us to explicitly fix the TAM width for each core to one of its Pareto-optimal points, thereby eliminating wasteful TAM wires; see Fig. 2. The staircase nature of the testing time variation with TAM width for cores is thus exploited to reduce the TAM width assigned to cores to the minimal value required to achieve a specific testing time. The extra TAM wires can be used for other cores in the SOC, thereby reducing total testing time. Furthermore, it is even possible to assign the entire TAM width to a certain core if required. (For fixed-width test buses, assigning the entire TAM width to a core is seldom efficient since there can be only one TAM and all cores receive the complete TAM width.) Moreover, in the new approach, TAMs can fork and merge between cores to improve TAM wire utilization. TAMs that fork and merge can also ultimately lead to lower TAM wiring area if implemented carefully. This is because all the wires of a wide TAM that are connected to a large core need not be routed to other smaller cores that are also on the same TAM. Finally, in fixed-width architecture design, test scheduling is performed after TAM design, by shifting tests back and forth on the TAMs to avoid conflicts. In flexible-width architecture design, however, test scheduling is tightly integrated with TAM design, leading to a more effective overall test planning flow.

4 Integrated TAM Design and Test Scheduling

The general integrated wrapper/TAM cooptimization and test scheduling problem that we address in this paper is as follows: We are given the total SOC TAM width W and the test set parameters for each core, i.e., the numbers of input, output, and bidirectional terminals, test patterns, scan chains, and the scan chain lengths. Unlike in [1], we assume that the scan chains in the cores cannot be redesigned, i.e., the number and lengths of scan chains are fixed. The goal is to determine the TAM width and a wrapper design for each core and a test schedule that minimizes the testing time for the SOC such that the following constraints are satisfied.

1. The total number of TAM wires utilized at any moment does not exceed W;
2. Precedence constraints are met;
3. Concurrency constraints are met;
4. The maximum power consumption value is not exceeded during test;
5. Selective preemption of tests is allowed.

An additional goal is to determine a value of W for the SOC to trade off testing time with tester data volume.

The overall optimization problem consists of three main parts: wrapper and TAM cooptimization, test scheduling, and identification of a TAM width for tester data volume reduction. These parts must be solved in conjunction to achieve the minimum system testing time and reduced tester data volume. We formulate a progression of three problems of increasing complexity that lead up to the overall optimization problem. These three problems are as follows:

Problem 1. Wrapper/TAM cooptimization and test scheduling.

Problem 2. Wrapper/TAM cooptimization and test scheduling with selective preemption and precedence and power constraints.

Problem 3. Wrapper/TAM cooptimization, test scheduling with selective preemption, and precedence and power constraints, and identification of a TAM width to trade off testing time with tester data volume.

In this section, we address Problem 1 and show how wrapper/TAM cooptimization can be integrated with test scheduling. In Section 5, we show how this problem is generalized to include precedence, preemption, and power-constraints—Problem 2. Finally, in Section 6, we study Problem 3, i.e., we identify TAM widths that provide a trade off between test time and tester data volume.

Problem 1. Given the test set parameters for each core, and the total TAM width W for the SOC, determine the TAM width and a wrapper design for each core and a test schedule for the SOC that minimizes the total testing time such that the total number of TAM wires utilized at any moment does not exceed W.

We solve the problem of wrapper design for cores using the *Design_wrapper* algorithm [13] based on the Best Fit Decreasing heuristic for the Bin Packing problem. In order to solve the problem of assigning TAM width to cores and scheduling tests, we represent core tests by rectangles and develop a rectangle packing algorithm. The use of rectangles for core test representation during test scheduling has been previously studied in [6], [11], [17]. The *Design_wrapper* algorithm is used to obtain the different test application times for each core for varying values of TAM width. A set of rectangles representing these different testing times for each value of TAM width for a core can now be constructed such that the height of the rectangle corresponds to the TAM width and the width of the rectangle represents the core test application time for this value of TAM width.

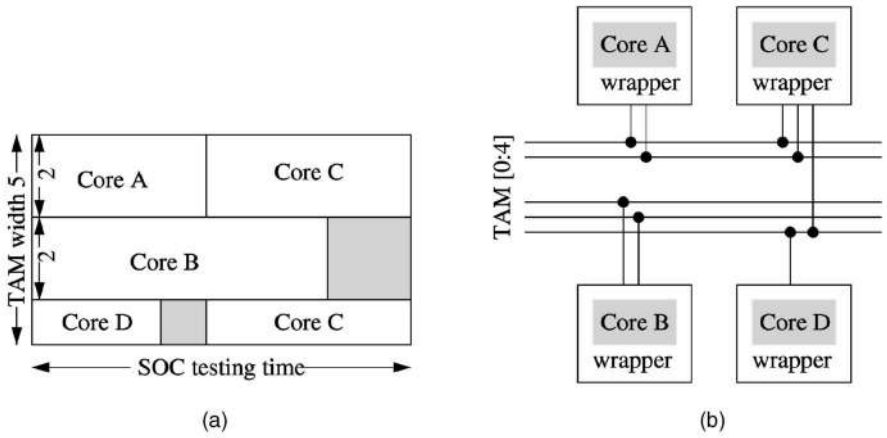

Fig. 3. An illustration of (a) rectangle splitting and (b) the corresponding TAM architecture.

We now formulate Problem 1 as a generalized version of the rectangle packing problem [7]. The rectangle packing problem, termed $\mathcal{P}_{\mathrm{RP}}$, is described as follows: Given a collection of rectangles and a bin of fixed height and unbounded width, pack the rectangles into the bin such that no two rectangles overlap and the width to which the bin is filled is minimized.

Problem $\mathcal{P}_{\mathrm{RP}}$ is extended to address Problem 1 in two ways. First, we are given a collection of *sets* of rectangles and one rectangle must be chosen from each set for packing. Second, during packing, each rectangle chosen can be vertically split into several nonadjacent rectangles having the same width. The generalized version of $\mathcal{P}_{\mathrm{RP}}$ that addresses Problem 1 is termed $\mathcal{P}_{\mathrm{GRP}}1$ and is formulated as follows: Consider an SOC having $|\mathbf{C}|$ cores (where $\mathbf{C}$ is the set of cores) and let $\mathbf{R}_i$ be the set of rectangles for core i, $1 \leq i \leq |\mathbf{C}|$. Problem $\mathcal{P}_{\mathrm{GRP}}1$ (generalized rectangle packing) is stated as follows:

$\mathcal{P}_{\mathrm{GRP}}1$**.** Select one rectangle $R_{ij} \in \mathbf{R}_i$ from each set $\mathbf{R}_i$, $1 \leq i \leq |\mathbf{C}|$, and pack the selected rectangles into a bin of fixed height and unbounded width such that no two rectangles overlap and the width to which the bin is filled is minimized. Each rectangle selected is allowed to be split vertically into several nonadjacent pieces, each having the same coordinates on the horizontal axis.

In Problem $\mathcal{P}_{\mathrm{GRP}}1$, during packing, the rectangle selected for a core can be vertically split into several non-adjacent rectangles having the same width, as illustrated for Core C in Fig. 3a. This is because it is possible to assign a group of noncontiguous TAM wires to a single core, using fork-and-merge of TAM wires, as illustrated in Fig. 3b. All the pieces of the split rectangle must, however, have the same coordinates on the horizontal axis. Recall that, in [7], rectangles are considered to be indivisible entities.

The relationship between Problem $\mathcal{P}_{\mathrm{GRP}}1$ and Problem 1 is illustrated in Fig. 4. The height of the rectangle selected for a core corresponds to the TAM width assigned to the core, while the rectangle width corresponds to the testing time for the core. The height of the bin corresponds to the total SOC TAM width and the width to which the bin is ultimately filled corresponds to the SOC testing time. The unfilled area of the bin corresponds to the idle time on TAM wires during test. Furthermore, the distance between the left edge of each rectangle and the left edge of the bin corresponds to the beginning time of each core test. Thus, a one-to-one correspondence exists between the packed bin and the final test schedule.

Problem $\mathcal{P}_{\mathrm{GRP}}1$ can be shown to be $\mathcal{NP}$-hard by a restriction argument. A special case of $\mathcal{P}_{\mathrm{GRP}}1$ in which the

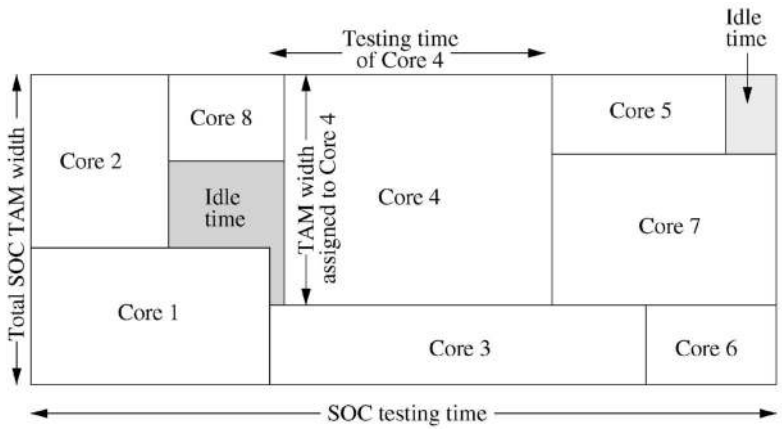

Fig. 4. Example test schedule using rectangle packing.

cardinality of each set $\mathbf{R}_i$, $1 \leq i \leq |\mathbf{C}|$, equals one directly corresponds to the rectangle packing problem in [7]. Since the rectangle packing problem was shown to be $\mathcal{NP}$-hard in [7] (by restriction to Bin Packing), $\mathcal{P}_{\mathrm{GRP}}1$ is also $\mathcal{NP}$-hard.

5 Constraint-Driven Test Scheduling

In this section, we first detail Problem 2 (integrated TAM design and constraint-driven test scheduling) and then formulate Problem $\mathcal{P}_{\mathrm{GRP}}2$, a generalized version of Problem $\mathcal{P}_{\mathrm{GRP}}1$ that is equivalent to Problem 2.

Problem 2. Given the test set parameters for each core, and the total TAM width W for the SOC, solve Problem 1 such that

1. precedence constraints are met,
2. concurrency constraints are met,
3. the maximum power consumption value P_{max} is not exceeded, and
4. selective preemption of tests is allowed.

Precedence constraints are intended to express user-defined (partial) ordering constraints. These arise due to interdependencies between various core tests. For example, a certain Core A might be tested before Core B because the test of Core B relies on the integrity of Core A. Furthermore, precedence constraints can be introduced such that tests which are more likely to fail are scheduled before tests that are less likely to fail [15]. Even though precedence constraints can increase the overall test completion time, "abort-at-first-fail" test strategies potentially lead to a reduced average test time. Next, concurrency constraints seek to avoid conflicts in the test hardware. For example, a hierarchical parent core cannot be tested at the same time as the child cores lying within it. This is because the wrappers of the child cores must be in Extest (External Test) mode while the parent core is being tested in Intest (Internal Test) mode. Finally, power constraints must be incorporated in the schedule to ensure that the power budget of the SOC is not exceeded during test [6].

Problem 2 can be expressed in terms of rectangle packing as follows: Consider an SOC having $|\mathbf{C}|$ cores, and:

1. Let $\mathbf{R}_i$ be the set of rectangles for core i, $1 \leq i \leq |\mathbf{C}|$;
2. Let precedence constraints between tests be defined, e.g., $i < j$ (test i must complete before test j is begun);
3. Let concurrency constraints between tests be defined, e.g., $i <> j$ (test i must not be applied at the same time as test j);
4. Let the test for Core i have a power consumption of P_i;
5. Let Core i be assigned a maximum number of allowed preemptions $max\_preempts(i)$.

$\mathcal{P}_{\mathrm{GRP}}2$ Select one rectangle R_{ij} from each set $\mathbf{R}_i$ and pack the selected rectangles into a bin of fixed height and unbounded width such that the bin width is minimized. Each rectangle selected is allowed to be split vertically into several nonadjacent pieces, with each piece having the same coordinates on the horizontal axis, such that the core test can occupy noncontiguous TAM wires. In addition, to introduce preemption of tests, each rectangle can be partitioned into several nonadjacent rectangles along the horizontal (time) axis. The number of these horizontal splits for a rectangle must not exceed the maximum number of preemptions allowed for the test. For each precedence constraint $i < j$, the rectangle for Core j can be packed only after all partitions of the rectangle for Core i are packed. Furthermore, the rectangle representing test i must not overlap (in time) the rectangle for test j, if there is a specified concurrency constraint $i <> j$. Finally, at any moment of time, the sum of the P_i values for the rectangles selected must not exceed the maximum specified value P_{max}.

Problem $\mathcal{P}_{\mathrm{GRP}}2$ is a generalized version of $\mathcal{P}_{\mathrm{GRP}}1$ and can therefore be shown to be $\mathcal{NP}$-hard by a restriction argument.

The value of P_i is supplied by the core vendor for each Core i, while the value of $max\_preempts(i)$ for each core is decided by the system integrator, allowing extra flexibility. For example, the system integrator may decide not to preempt BIST tests or sequential circuit tests to avoid having to store the states of core flip-flops and LFSRs. Thus, $max\_preempts(i)$ can be set to 0 for such cores.

Next, we describe the algorithm that we use to solve Problem $\mathcal{P}_{\mathrm{GRP}}2$, and demonstrate how the relationship between TAM width and testing time is exploited to maximize TAM wire utilization during constraint-driven test scheduling. For each Core i, our algorithm first identifies a "preferred TAM width," $width_p(i)$, such that, at $width_p(i)$, the core's testing time is close to its lowest value achievable at the maximum allowable TAM width W_{max}. (In this paper, W_{max} is chosen to be 64.) A group of tests is selected to be scheduled such that precedence and power constraints are met and test conflicts are avoided. The algorithm uses heuristics that seek to insert tests into the available time on TAM wires. If idle time is inevitable, the algorithm waits until the first currently running test completes and then repeats the scheduling process for the remaining tests. Tests may be preempted and resumed again such that the number of preemptions for any Core i does not exceed $max\_preempts(i)$. Next, we explain the rationale behind the heuristic decision-making in our algorithm and show how these decisions minimize system testing time.

Data structure. The data structure in which we store the TAM width and testing time values for the cores of the SOC is presented in Fig. 5. This data structure is updated with the assigned TAM width, begin and end times, and preemption count for each core as the test schedule is developed.

The pseudocode for our TAM optimization and test scheduling algorithm is presented in Fig. 6. The inputs to our algorithm are the set **C** of cores, total TAM width W, set PC of precedence constraints, set **CC** of concurrency constraints, power constraint P_{max}, and user-input parameters p and d (explained later).

Preferred TAM widths. In Procedure *Initialize* (Line 1), we calculate the collection **R** of Pareto-optimal rectangles, and the preferred TAM width values for each core from the

Data structure *Schedule*

1 $width_p(i)$ /* preferred TAM width for Core i */
2 $width(i)$ /* TAM width assigned to Core i */
3 $first\_begin(i)$ /* first begin time of Core i */
4 $end(i)$ /* end time of Core i */
5 $scheduled\_times(i)$ /* list of times during which Core i is scheduled */
6 $time\_left(i)$ /* testing time remaining for Core i */
7 $begun(i)$ /* boolean indicates Core i has begun at least once */
8 $scheduled(i)$ /* boolean indicates Core i is scheduled */
9 $complete(i)$ /* boolean indicates test for Core i has completed */
10 $preempts(i)$ /* number of times Core i has been preempted */
11 $max\_preempts(i)$ /* maximum number of preemptions allowed */

Fig. 5. Data structure for the test schedule.

input percent value p. Recall that the core testing time varies with TAM width w as a staircase function that drops rapidly at first for small values of w and less rapidly after that. For example, for Core 6 in p93791 (Fig. 1b), at $w = 10$, the testing time reaches within 10 percent of its value at $w = 64$, and, at $w = 15$, the testing time is within 5 percent of its value at $w = 64$. However, the highest Pareto-optimal value of TAM width is $w = 47$. Hence, instead of attempting to assign the highest Pareto-optimal width to a core, a considerable savings in system TAM width can be realized by assigning a precalculated preferred value of width such that the testing time of the core reaches within a small percent value p of its testing time at $w = W_{max}$. The value of p is usually between 1 and 10. While we will attempt to pack only the rectangle representing the preferred TAM width for each core, the other rectangles in the set remain candidates for packing. This is because, in the event that a

Procedure *TAM_schedule_optimizer*($\mathbf{C}$, W, $\mathbf{PC}$, $\mathbf{CC}$, P_{max}, d, p)

1 *Initialize*($\mathbf{C}$, d, p);
2 Set $w\_avail = W$; $current\_time = 0$;
3 **While** $\mathbf{C} \neq \emptyset$
4 **If** $w\_avail > 0$
5 **If** a Core $i \in \mathbf{C}$ can be found, such that $begun(i) < current\_time$ AND $scheduled(i) = 0$ AND $preempts(i) = max\_preempts(i)$
6 *Assign*($i, width(i), 0$);
7 **If** a Core $i \in \mathbf{C}$ can be found, such that $begun(i) = 1$ AND $width(i) \leq w\_avail$ AND $time\_left(i)$ is maximum AND (*Conflict*(i, $\mathbf{PC}$, $\mathbf{CC}$, P_{max})=0)
8 **If** $end(i) < current\_time$
9 *Assign*($i, width(i), 1$);
10 **Else** *Assign*($i, width(i), 0$);
11 **If** a Core $i \in \mathbf{C}$ can be found, such that $begun(i) = 0$ AND $width_p(i) \leq w\_avail$ AND $time\_left(i)$ is maximum AND (*Conflict*(i, $\mathbf{PC}$, $\mathbf{CC}$, P_{max})=0)
12 *Assign*($i, width_p(i), 0$);
13 **If** a Core $i \in \mathbf{C}$ can be found, such that $begun(i) = 0$ AND $width_p(i) \leq w\_avail + 3$ AND $width_p(i)$ is minimum AND (*Conflict*(i, $\mathbf{PC}$, $\mathbf{CC}$, P_{max})=0)
14 *Assign*($i, w\_avail, 0$);
15 **If** a Core $i \in \mathbf{C}$ can be found, such that $first\_begin(i) = current\_time$ AND $T_i(width(i)) - T_i(width(i) + w\_avail)$ is maximum;
16 *Assign*($i, width(i) + w\_avail, 0$);
17 **Else** *Update*($\mathbf{C}$);
18 **Return** *Schedule*;

Fig. 6. Algorithm for solving $\mathcal{P}_{rmGRP}2$.

Procedure *Initialize*($\mathbf{C}$, d, p)

1 Compute collection $\mathbf{R}$ of rectangles using *Design_wrapper*;
2 **For** each Core $i \in \mathbf{C}$
3 Calculate $T_{ip} = T_i(W_{max}) + \frac{p}{100} \times (T_i(1) - T_i(W_{max}))$;
4 Set $width_p(i) = w$, such that $T_i(w) - T_{ip}$ is minimum;
5 Calculate highest Pareto-optimal width w_h;
6 **If** $w_h - width_p(i) \leq d$ **then** $width_p(i) = w_h$;

Fig. 7. Preferred widths initialization subroutine.

preferred rectangle introduces empty space in the bin, a nonpreferred rectangle from the set can be packed instead.

In subroutine *Initialize* (Fig. 7), we initialize $width_p(i)$ to the Pareto-optimal TAM width that provides the closest testing time to the calculated value of time T_{ip} within p percent from $T_i(W_{max})$. (We use $T_i(w)$ to denote the testing time of Core i when provided with a TAM width of w.) In Lines 5 and 6 of *Initialize*, we make an allowance for $width_p(i)$ to be set to the highest Pareto-optimal TAM width w_h if the difference between the value of $width_p(i)$ from Line 3 and the value of w_h is less than the input difference value d. This heuristic aids significantly in minimizing system testing time, especially when it is beneficial to assign a few ($\leq d$) extra TAM wires to a bottleneck core in the system. For example, when using $p = 2$ for benchmark SOC p34392 [21] (presented in Section 7), we noticed that Core 18 was assigned $width_p(18) = 9$ bits, leading to a testing time of $T_{18}(9) = 622,163$ cycles. The testing time for the SOC was also found to be 622,163 cycles, from which we noted that Core 18 is a bottleneck core for p34392. A further study of the testing time-TAM width characteristics of Core 18 revealed that its highest Pareto-optimal TAM width is 10 bits, at which the testing time for Core 18 reaches its minimum value of 544,579 cycles. Hence, providing an extra TAM wire to Core 18 reduced its testing time as well as the overall SOC testing time to $T_{18}(10) = 544,579$ cycles. Thus, the minimum testing time for SOC p34392 could be achieved using the heuristic in Lines 5 and 6 of *Initialize* with $d = 1$.

TAM width assignment and test scheduling. Line 2 of *TAM_schedule_optimizer* initializes the main rectangle packing loop. While executing the main **While** loop (Line 3), if there are w_avail ($1 \leq w\_avail \leq W$) TAM wires available for assignment, cores are assigned to the TAM using a three-priority selection mechanism:

Priority 1: Line 5 searches for a Core i whose test has already been preempted the maximum allowable number of times ($max\_preempts(i)$), but has not completed. If such a core is found, it is scheduled using the *Assign* subroutine shown in Fig. 8. After *Assign* completes, control is returned to Line 4 of *TAM_schedule_optimizer* and the process of selecting a core begins again. Thus, only if no core is found by *Priority* 1 does *Priority* 2 come into play.

Priority 2: If a core is found whose test has begun earlier whose assigned TAM width is less than or equal to w_avail and whose remaining testing time is largest among all such cores, then it is scheduled using *Assign* in Lines 7 to 10.

Procedure *Assign*$(i, width, preempt)$

1 Let i be the core to be scheduled;
2 Set $width(i) = width$; $w\_avail = w\_avail - width$;
3 Set $scheduled(i) = 1$; $preempts(i) = preempts(i) + preempt$;
/*s_i, s_o are the longest wrapper scan-in and scan-out lengths [13]*/
5 **If** $preempt = 1$, Set $time\_left(i) = time\_left(i) + \min\{si, so\}$;
6 **If** $begun(i) = 0$
7 Set $begun(i) = 1$; $first\_begin(i) = current\_time$;
9 Set $time\_left(i) = T_i(width(i))$;
10 Set $end(i) = current\_time + time\_left(i)$;
11 **Return** to Line 4 of *TAM_schedule_optimizer*;

Fig. 8. The core assign algorithm.

Procedure *Conflict*$(i,$ **PC**, **CC**, $P_{max})$

1 Let i be the core to be scheduled;
2 **For** all precedence constraints $j < i$
3 **If** $complete(j) = 0$, **Return** 1;
4 **For** all concurrency constraints $i <> j$
5 **If** $scheduled(j) = 1$, **Return** 1;
6 Let $P = 0$;
7 **For** all cores $j \neq i$
8 **If** $scheduled(j) = 1$, $P = P + P_j$;
9 **If** $P > P_{max}$, **Return** 1;
10 **For** all cores $j \neq i$
11 **If** there is a BIST–scan test conflict between Cores i and j, **Return** 1;
12 **Return** 0;

Fig. 9. Precedence, concurrency, and power constraints.

Priority 3: If a core is found whose test has not begun earlier whose preferred TAM width is less than or equal to w_avail and whose remaining testing time is largest among all such cores, then it is scheduled using *Assign* in Lines 11 to 12.

Priority 1 is motivated by the need to complete the test for cores that cannot be preempted further, while *Priorities* 2 and 3 seek to assign the preferred TAM width to each core.

Precedence, concurrency and power constraints. During selection of a core to be scheduled in Lines 7, 11, and 13 of *TAM_schedule_optimizer*, the *Conflict* subroutine (Fig. 9) is invoked to ensure that 1) precedence conflicts, 2) concurrency conflicts, and 3) power constraint conflicts are avoided.

Rectangle insertion in idle time. If there is no core found in Lines 5 to 12, rather than let the w_avail TAM wires remain idle, *TAM_schedule_optimizer* attempts to insert the rectangle for some unscheduled core into the available time. In Line 13, we find a core that has not been scheduled and whose preferred TAM width is less than or equal to $w\_avail + 3$. This core is then scheduled using *Assign*. The 3-bit limit was found to be the most effective after extensive experimentation. We have observed that, as long as the difference between the width of the chosen rectangle and the preferred width is less than 3, rectangle insertion leads to substantial reduction in testing time. However, if a value different from 3 is found to be more useful for a different set of SOCs, the new value can be readily provided as an input by the system integrator during test automation.

Increasing TAM widths to fill idle time. If no rectangle is available to fill in the idle time, then the heuristic in Lines 15 to 16 is used to determine which of the cores currently scheduled to begin at $current\_time$ will benefit the most, in terms of testing time decrease, from an extra w_avail TAM wire. If such a core can be found, then its currently assigned $width(i)$ TAM wires are increased to the highest Pareto-optimal width less than $width(i) + w\_avail$. This heuristic is illustrated in Fig. 10. In Fig. 10, after Cores 2 and 3 have been assigned their preferred widths, no other core's preferred width is small enough to fit in the idle time above the rectangle for Core 3. Furthermore, there is no core available for which a rectangle can be made to fit in the idle time. Therefore, Core 2 is selected to have its width increased to $width(2) + w\_avail$. Note from Fig. 10 that the width of a core (e.g., Core 1) whose begin time lies before $this\_time$ cannot be increased at $this\_time$ because its test has already begun with $width(i)$ TAM wires. This requirement for Core i is ensured by the first argument to the AND condition in Line 15.

Finally, if the heuristics in Lines 4 to 16 fail to find a core to assign, the value of w_avail is set to 0 and the loop beginning at Line 4 is repeated. When w_avail is found to be 0 in Line 4, the execution proceeds to Line 17, where the process of updating $current\_time$ and w_avail is begun. This is presented in subroutine *Update* in Fig. 11. Once $current\_time$ is incremented, the widths assigned to packed rectangles (or parts of rectangles) are fixed and cannot be changed later on in the schedule. The resulting test schedule is output in Line 18.

Test preemption. Tests can be selectively preempted each time *Update* is executed. At each execution of *Update*, the value of w_avail is reset to W and all the incomplete tests contend for the available TAM width in Lines 4 to 16 of *TAM_schedule_optimizer*. If the maximum limit on preemptions for a certain Core i is reached in Line 5, then Core i is continuously scheduled until it completes. On the other hand if the limit on preemptions is not reached and Core i is

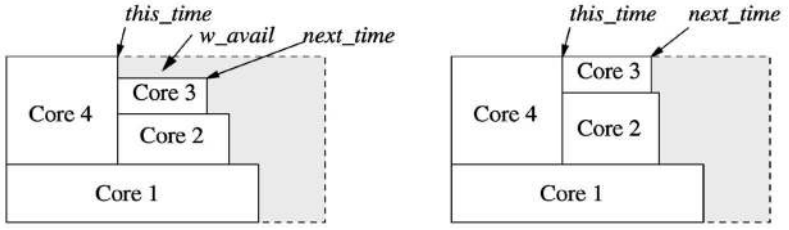

Fig. 10. Increasing TAM width to fill idle time.

Procedure *Update*($\mathbf{C}$)

1 Find Core i such that $time\_left(i)$ is minimum;
2 Set $next\_time = current\_time + time\_left(i)$;
3 **For** all Cores i such that $scheduled(i) = 1$
4 Set $scheduled(i) = 0$; $end(i) = next\_time$;
5 Set $time\_left(i) = time\_left(i) - (next\_time - current\_time)$;
6 **If** $time\_left(i) = 0$
7 Set $complete(i) = 1$; $\mathbf{C} = \mathbf{C} - i$;
8 Update $scheduled\_times(i)$;
9 Set $current\_time = next\_time$; $w\_avail = W$;

Fig. 11. Test schedue update algorithm.

preempted (Line 9), then $time\_left(i)$ is incremented by $\min\{s_i, s_o\}$ in Line 5 of *Assign*. Here, s_i (s_o) is the length of the longest wrapper scan in (out) chain [19]. This increment in testing time is because each time a preemption occurs, an extra scan in or scan out must be performed. This is explained as follows: Let $\mathcal{T} = (\max\{s_i, s_o\} + 1) \cdot p + \min\{s_i, s_o\}$ be the testing time for a core [20], where p is the number of test patterns. Let the test for the core be preempted t times. Therefore, let $p = p_1 + p_2 + \ldots + p_{t+1}$. We have

$$\mathcal{T} = (\max\{s_i, s_o\} + 1) \cdot p_1 + \min\{s_i, s_o\} + \ldots (\max\{s_i, s_o\} + 1) \cdot p_{t+1} + \min\{s_i, s_o\}.$$

This yields $\mathcal{T} = (\max\{s_i, s_o\} + 1) \cdot p + (t+1) \cdot \min\{s_i, s_o\}$. The increase in testing time is therefore $t \cdot \min\{s_i, s_o\}$ cycles.

The complexity of the rectangle packing algorithm can be estimated as follows. The **While** loop in Line 3 of Fig. 6 is executed $|\mathbf{C}|$ times, where $|\mathbf{C}|$ is the number of cores of the SOC. In each such execution, the set of unassigned cores is searched using a linear search, i.e., $\mathcal{O}(|\mathbf{C}|$ cores are examined as potentially schedulable (Lines 5, 7, 11, 13, 15 of Fig. 6). During these linear searches, for each unassigned core examined, all currently scheduled cores ($\mathcal{O}(|\mathbf{C}|)$ are also examined using the *Conflict* subroutine to determine whether there are any precedence or power conflicts. From this, it follows that the complexity of scheduling is $\mathcal{O}(|\mathbf{C}|^3)$. Note that we do not include the complexity of generating the rectangles here. The complexity of rectangle generation using *Design_wrapper* is $\mathcal{O}(sc \log sc + sc \cdot k)$, where sc is the number of internal scan chains in the cores of the SOC and k is the TAM width [13].

6 Tester Data Volume Reduction

In the previous sections, we presented the first two parts of our framework: wrapper/TAM cooptimization and test scheduling to minimize testing time, given a total TAM width value W for the SOC. In this section, we present the third part of our framework—the identification of a value of W ($W \leq W_{max}$) that minimizes a weighted cost function involving both the testing time and the tester data volume, where W_{max} is the maximum allowed TAM width for the SOC.

We first motivate the need for identifying such a TAM width. The cost of testing SOCs is closely related to the testing time and the volume of test data. While the time required to apply digital patterns to the SOC is a relatively small fraction of the total test time, the time required to transfer several Gigabytes of data from a workstation to the tester memory is significant if performed frequently [2]. Techniques to ensure that the test data required per pin is contained to a single tester buffer are therefore vital. The motivation for trading off TAM width with testing time and data volume lies in multisite testing in which several ICs are tested in parallel by a single tester. Reduced TAM widths that do not increase test data per pin beyond buffer sizes will allow a larger number of ICs to be tested concurrently, thereby decreasing testing time for the entire production batch. Reducing TAM widths also leads to lower routing complexity on-chip. It is therefore important to develop techniques that can identify a low number of TAM wires (scan data buffers) and to trade off testing time and data volume.

We begin by plotting the variation of testing time $\mathcal{T}$ with W. This is shown for benchmark SOC p21241 [21] in Fig. 12a. Note the decrease in $\mathcal{T}$ with increasing W. Next, we plot the variation of tester data volume $\mathcal{M}$ with W (Fig. 12b). $\mathcal{M}$ varies as a nonmonotonic function in W, achieving local minima at the Pareto-optimal W values of the $\mathcal{T}$ curve of Fig. 12a. The global minimum (marked in Fig. 12b) is achieved at $W = 44$. However, $W = 44$ does not provide the lowest testing time for the SOC. Testing time $\mathcal{T}$ can be decreased by increasing W from 44 to 48 (at which point there is an increase in $\mathcal{M}$). Therefore, by varying W, the system integrator can trade off testing time with tester data volume.

We incorporate this feature in our framework by defining the normalized cost function

$$\mathcal{C} = \alpha \frac{\mathcal{T}}{\mathcal{T}_{min}} + (1 - \alpha) \frac{\mathcal{M}}{\mathcal{M}_{min}},$$

where $\mathcal{T}_{min}$ ($\mathcal{M}_{min}$) is the minimum value of $\mathcal{T}$ ($\mathcal{M}$) and $0 \leq \alpha \leq 1$ is a user-input parameter to control the trade off. As α is varied from 0 to 1, the shape of the $\mathcal{C}$-curve changes from the $\mathcal{M}$-curve to the $\mathcal{T}$-curve. The $\mathcal{C}$-curve is "U" shaped in general, having a single minima, illustrated for $\alpha = 0.5$ in Fig. 13a and for $\alpha = 0.75$ in Fig. 13b. These values of W that minimize $\mathcal{C}$ for various values of α provide the system integrator with effective choices of TAM width for tester data volume reduction.

7 Experimental Results

In this section, we present experimental results for four ITC 2002 SOC test benchmarks [21]. The experimental results were obtained using a Sun Ultra 10 workstation with a 333 MHz processor and 256 MB memory.

Table 1 presents results of integrated wrapper/TAM cooptimization and test scheduling for SOCs d695 and p22810. No precedence or power constraints were included in this set of experiments. We considered all possible integer values of the parameters p and d in the range $1 \leq p \leq 10$, $0 \leq d \leq 4$, and tabulated the best results. The symbols $\mathcal{T}$ and $\mathcal{E}$ are used to represent the SOC testing time (expressed in clock cycles) and the computation time (expressed in seconds), respectively, of the ILP/enumeration-based algorithm [13]. The symbol $\mathcal{T}_{new}$ represents the SOC testing time (expressed in clock cycles) of the new rectangle packing algorithm. The

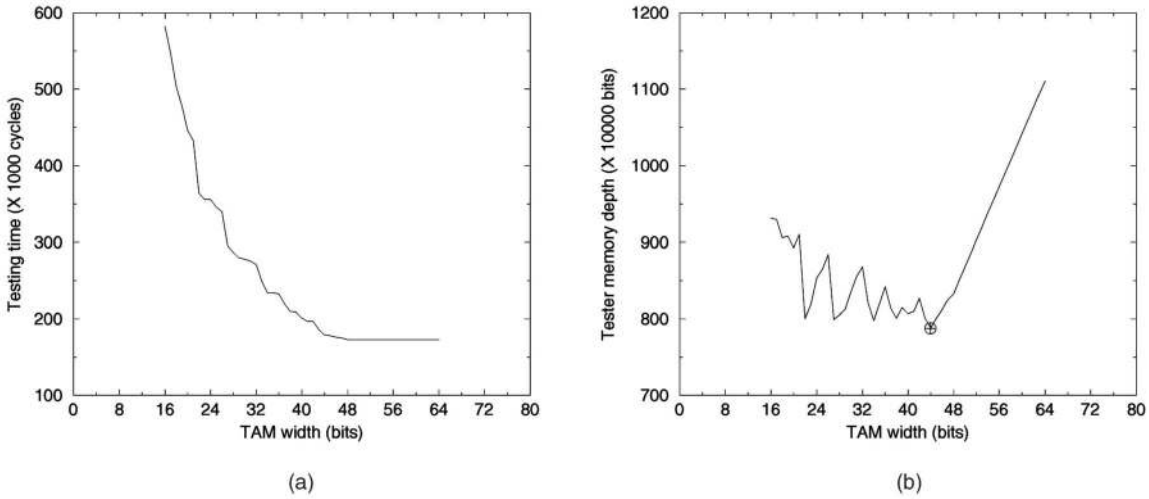

Fig. 12. Relationship between (a) $\mathcal{T}$ and W and (b) $\mathcal{M}$ and W for SOC p21241.

percentage change in testing time using the new method is calculated using the formula $\Delta\mathcal{T}$ (percent) $= \frac{\mathcal{T}_{new}-\mathcal{T}}{\mathcal{T}} \times 100$. The computation time of the new algorithm is less than 1 second in each case; hence, these times are not mentioned in Table 1. (Note that the CPU time for the ILP solver used in [13] does not increase monotonically with W.)

The testing times for d695 obtained using the proposed method are comparable to the testing times obtained using the ILP/enumeration-based method in [13]. For p22810, however, the new method yields a significantly lower SOC testing time. This is because the problem instance size is larger and SOC p22810 has a larger number of cores; thus, our rectangle packing heuristics have more room for rectangle manipulation and height-width optimization. The values of $\mathcal{E}$ shown for p22810 in Table 1 are for two TAMs. This is because the ILP models for p22810 were particularly intractable and the ILP method [13] did not run to completion for three or more TAMs, even after two days of execution. The CPU time of our new algorithm is several orders of magnitude lower than the CPU times required by the method in [13]; the execution speed-up factor can, in fact, be estimated from the values of $\mathcal{E}$ in Table 1 since the new algorithm takes less than 1 second to execute in each case.

Table 2 presents results for SOCs p34392 and p93791. The values of $\mathcal{T}$ shown for p34392 are for three TAMs. (For four TAMs, the ILP method of [13] did not provide a solution, even after two days of CPU time.) For p34392, we reach the optimum (lower bound) testing time of 544,579 cycles at $W = 32$. This lower bound corresponds to the time taken to test the bottleneck core, Core 18, when it is supplied with a TAM width equal to its highest Pareto-optimal point. An expression for this lower bound on the system testing time for an SOC was derived in [4]. The ILP/enumeration-based method requires a total TAM width of 40 to reach this lower bound. Note that the values of $\mathcal{E}$ do not increase monotonically with W because the time taken by the ILP tool to solve the $\mathcal{NP}$-hard problems in [13] varies significantly with the problem instance; different width partitions for the same W value result in widely varying CPU times. In Fig. 14, we present the test schedules obtained for p34392

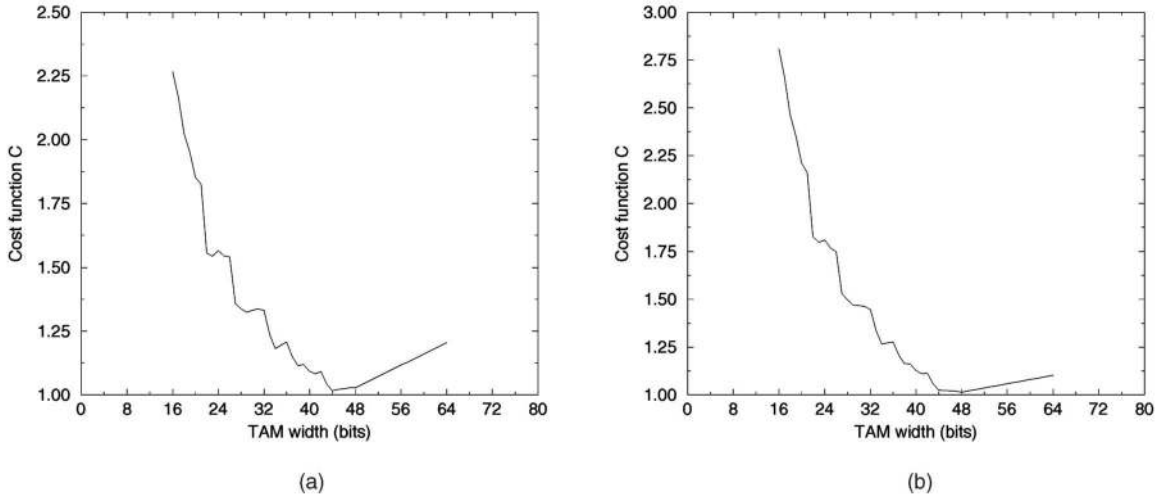

Fig. 13. The normalized cost function $\mathcal{C}$ for (a) $\alpha = 0.5$ and (b) $\alpha = 0.75$ for SOC p21241.

TABLE 1
Results for d695 and p22810

	SOC d695				SOC p22810			
	Method of [13]: $B = 3$		New method		Method of [13]: $B = 2$		New method	
W	$\mathcal{T}$ (cc)	$\mathcal{E}$ (sec)	$\mathcal{T}_{new}$ (cc)	$\Delta\mathcal{T}$ (%)	$\mathcal{T}$ (cc)	$\mathcal{E}$ (sec)	$\mathcal{T}_{new}$ (cc)	$\Delta\mathcal{T}$ (%)
16	42568	16	43723	+2.71	462210	11	452639	-2.07
24	28292	40	30317	+7.16	361571	24	307780	-14.88
32	21566	67	23021	+6.75	312659	49	246150	-21.27
40	17901	105	18459	+3.12	278359	60	197293	-29.12
48	16975	159	15698	-7.52	268472	84	167256	-37.70
56	13207	210	13415	+1.57	266800	80	145417	-45.50
64	12941	290	11604	-10.33	260638	122	136941	-47.46

TABLE 2
Results for p34392 and p93791

	SOC p34392				SOC p93791			
	Method of [13]: $B = 3$		New method		Method of [13]: $B = 3$		New method	
W	$\mathcal{T}$ (cc)	$\mathcal{E}$ (sec)	$\mathcal{T}_{new}$ (cc)	$\Delta\mathcal{T}$ (%)	$\mathcal{T}$ (cc)	$\mathcal{E}$ (sec)	$\mathcal{T}_{new}$ (cc)	$\Delta\mathcal{T}$ (%)
16	998733	222	1023820	+2.51	1771720	25	1851135	+4.48
24	720858	325	759427	+5.35	1187990	50	1248795	+5.12
32	591027	1576	544579	-7.86	887751	85	975016	+9.83
40	544579	1081	544579	0.00	698583	130	794020	+13.66
48	544579	6198	544579	0.00	599373	210	627934	+4.77
56	544579	11331	544579	0.00	514688	270	568436	+10.44
64	544579	1125	544579	0.00	460328	440	511286	+11.07

for $W = 32$ to further illustrate the difference between the way TAM width is allocated to cores by the method of [13] and by the new algorithm. The numbers between 1 and 19 in the rectangles of Fig. 14 denote cores.

The new testing times for p93791 (the largest example SOC having the most cores) are on average 8 percent higher than the testing times obtained using the method in [13]. A careful study of the test schedules of Table 2 and the number of I/Os, test patterns, and scan chain lengths of the cores in p93791 reveals that the vast differences between the test data requirements of the cores make it difficult for the proposed algorithm to optimize the system testing time using only a single value of p for all cores. We therefore added an additional heuristic to our algorithm to better allocate TAM resources to cores based on their test data volume. Rectangles for cores that have higher test data volume are packed using a lower value of p. Significantly, this new heuristic resulted in a further decrease in testing time for $W = 32$. The new values of $\mathcal{T}_{new}$ and $\Delta\mathcal{T}$ for $W = 32$ are as follows: $W = 32$: $\mathcal{T}_{new} = 940,916$, $\Delta\mathcal{T} = +5.99\%$. The new decreased testing time for $W = 32$ motivates further investigation into how the value of p affects testing time for each core and how it should be tailored to the test data needs of each individual core for larger SOCs such as p93791.

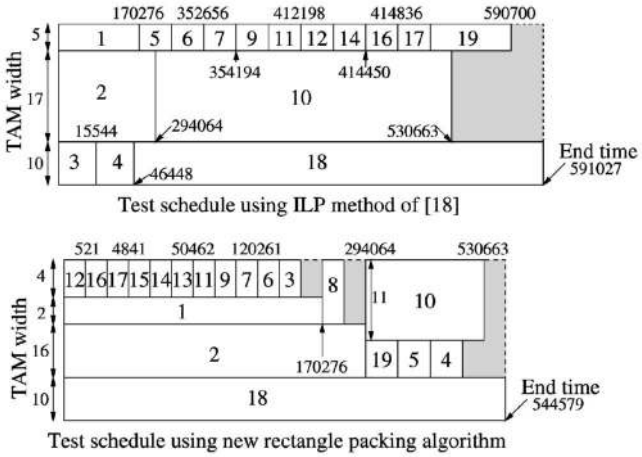

Fig. 14. Test schedules for p34392 using [13] and the new method (figures not drawn to scale).

Next, Table 3 presents results on integrated wrapper/TAM cooptimization and *constrained* test scheduling for the four example SOCs. We considered all possible integer values of the parameters p and d in the range $1 \leq p \leq 10$, $0 \leq d \leq 4$, and tabulated the best results. We first present a lower bound on the testing time for an SOC, given the test data for its cores.

Lower bound. A lower bound on the testing time for an SOC for total TAM width W is given by

TABLE 3
Wrapper/TAM Cooptimization and Test Scheduling

SOC	W	Testing time (cycles)			
		Lower bound	Non-preemptive	Preemptive	Preemptive+ power-constrained
d695	16	41232	43410	43423	47574
	32	20616	22229	21757	29039
	48	13744	15698	15499	28441
	64	10308	11285	11354	20004
p22810	16	421473	466383	459951	527573
	32	210737	243779	243978	277151
	48	140491	164420	162554	213845
	64	105369	140222	134732	176076
p34392	16	936882	1071043	1082065	1180187
	24	624588	728986	702322	1075971
	28	544579	617018	615126	1075242
	32	544579	544579	544579	1075242
p93791	16	1749388	1860752	1860752	1966092
	32	874694	929311	929311	1247221
	48	583130	637717	643605	656214
	64	437347	503661	492095	631840

$$\max\left\{\max_i T_i(W_{\max}), \left\lceil\left(\sum_{i=1}^{N} T_i(1)\right)/W\right\rceil\right\}.$$

Intuitively, this lower bound is derived by taking the greater of 1) the lower bound on testing time for the largest core and 2) the sum of the testing times of all the cores connected to a TAM of width 1 bit divided by W. Expression 2) assumes that no idle time is lost in the wrapper scan chains and that rectangle packing is perfect (no idle time on TAM wires). Lower bound values on the testing time for each SOC for several values of W are presented in Table 3. We compare the testing times obtained using nonpreemptive and preemptive scheduling. Note that the testing times obtained for nonpreemptive scheduling are slightly different from the testing time shown in Tables 1 and 2. This is because we used a simpler initial version of *TAM_schedule_optimizer* to obtain the test schedules presented in Tables 1 and 2. In this simpler version of the algorithm, rectangle insertion in idle-time was not performed. For preemptive testing, the value of $max\_preempts(i)$ (see Line 11 of Fig. 5) was set to 2 for the larger cores. Preemptive scheduling obtains lower or equal testing times in most cases. However, in a few cases, nonpreemptive schedules are shorter. This is because each preemption adds $\min\{s_i, s_o\}$ cycles to the length of a test, which can increase the overall SOC testing time for SOCs having a large number of short tests that are preempted several times. A careful investigation of the effects of preemption and the use of the $max\_preempts(i)$ parameter considering test lengths is therefore warranted. For p34392, we present testing time results only for $W \leq 32$. At $W = 32$, the testing time for p34392 reaches the lower bound of 544,579 cycles.

In Table 3, we also present the testing times obtained with power-constraints. We assigned a hypothetical power value P_i to the test for each Core i based on the number of test data bits per test pattern for Core i. The value of P_{max} was set to $\max_i\{P_i\}$ for test scheduling. Intuitively, this results in a test schedule in which the test with the maximum power consumption is not scheduled concurrently with any other test. The increases in testing times for power-constrained scheduling reflect this concurrency constraint. The CPU time of our new algorithm is several orders of magnitude lower than the CPU times required by the method in [13] since the new algorithm takes less than 1 second to execute in all cases.

Next, Table 4 presents results on identification of TAM widths for tester data volume reduction. We present the minimum values of $\mathcal{T}$ and $\mathcal{M}$ obtained for the four SOCs, as well as the values of W at which they occur. To illustrate the process of identifying TAM widths, we then present the values of $\mathcal{C}_{min}$ and the resulting effective TAM widths W_e obtained for several different values of α. Finally, the corresponding values of $\mathcal{T}$ and $\mathcal{M}$ obtained for these values of W_e are shown. It is clear from Table 4 that the system integrator can trade off testing time with tester data volume by varying α between 0 and 1. For example, for SOC p22810, the minimum value of $\mathcal{T}$ (140,222 cycles) is achieved at $W = 63$. However, the minimum values of $\mathcal{M}$ (7,377,480 bits) is actually achieved at $W = 44$. By setting $\alpha = 0.3$, the system integrator obtains a TAM width of 48 bits, at which $\mathcal{T} = 164,420$ cycles and $\mathcal{M} = 7,892,160$ bits.

TABLE 4
TAM Widths for Test Data Volume Reduction

SOC	$\mathcal{T}_{min}$ (cycles)	W at $\mathcal{T}_{min}$ (bits)	$\mathcal{M}_{min}$ (bits)	W at $\mathcal{M}_{min}$ (bits)	α	$\mathcal{C}_{min}$	W_e (bits)	$\mathcal{T}$ at W_e (cycles)	$\mathcal{M}$ at W_e (bits)
d695	11285	63	675554	22	0.1	1.031	60	11612	696720
					0.3	1.031	60	11612	696720
					0.5	1.026	63	11285	710955
p22810	140222	63	7377480	44	0.01	1.018	44	167670	7377480
					0.3	1.103	48	164420	7892160
					0.5	1.093	55	148005	8140275
p34392	544579	32	16659486	27	0.2	1.000	27	617018	16659486
					0.25	1.033	27	617018	16659486
					0.3	1.032	32	544579	17426528
p93791	503661	62	29399656	22	0.5	1.012	22	1336348	29399656
					0.95	1.000	62	503661	31226982
					0.99	1.000	62	503661	31226982

8 Conclusions

In this paper, we have presented a new technique based on rectangle packing for test wrapper and TAM cooptimization, test scheduling, and tester data volume reduction for SOCs. TAMs have been tailored to the test data needs of cores through the exclusive use of Pareto-optimal widths. We have also presented several novel heuristics that minimize the idle time on TAM wires, thereby leading to a fast and efficient algorithm for TAM width allocation and test scheduling. The new rectangle packing algorithm is scalable for large industrial SOCs and completes in less than 1 second of CPU time. This represents several orders of magnitude improvement over the exact methods for TAM optimization presented in earlier work. Finally, the proposed approach allows the system integrator to determine an SOC-level TAM width to trade off testing time with tester data volume.

Acknowledgments

The usage of Pareto-optimal TAM widths and rectangle packing was inspired by discussions with Jan Korst and Sandeep Koranne, respectively, for which the authors gratefully acknowledge them. They also thank Graeme Francis, Harry van Herten, and Erwin Waterlander for their help with providing data for the benchmark SOCs from Philips. Finally, they thank Harald Vranken and Sandeep Kumar Goel for their valuable comments on an earlier version of this paper. This research was supported in part by the US National Science Foundation under grants CCR-9875324 and CCR-0204077. Vikram Iyengar was supported by an IBM graduate fellowship. This work was performed while Vikram Iyengar was at Duke University. Preliminary versions of parts of this paper appeared in the *Proceedings of the VLSI Test Symposium*, pp. 253-258, April-May 2002, and the *Proceedings of the Design Automation Conference*, pp. 685-690, June 2002.

References

[1] J. Aerts and E.J. Marinissen, "Scan Chain Design for Test Time Reduction in Core-Based ICs," *Proc. Int'l Test Conf.*, pp. 448-457, 1998.
[2] C. Barnhart et al., "OPMISR: The Foundation for Compressed ATPG Vectors," *Proc. Int'l Test Conf.*, pp. 748-757, 2001.
[3] M. Benabdenbi, W. Maroufi, and M. Marzouki, "CAS-BUS: A Scalable and Reconfigurable Test Access Mechanism for Systems on a Chip," *Proc. Design, Automation, and Test in Europe (DATE) Conf.*, pp. 141-145, 2000.
[4] K. Chakrabarty, V. Iyengar, and A. Chandra, *Test Resource Partitioning for System-on-a-Chip.* Norwell, Mass.: Kluwer Academic, 2002.
[5] C.-A. Chen and S.K. Gupta, "Efficient BIST TPG Design and Test Set Compaction via Input Reduction," *IEEE Trans. Computer-Aided Design,* vol. 17, pp. 692-705, Aug. 1998.
[6] R.M. Chou, K.K. Saluja, and V.D. Agrawal, "Scheduling Tests for VLSI Systems under Power Constraints," *IEEE Trans. VLSI Systems,* vol. 5, no. 2, June 1997.
[7] E.G. Coffman Jr., M.R. Garey, D.S. Johnson, and R.E. Tarjan, "Performance Bounds for Level-Oriented Two-Dimensional Packing Algorithms," *SIAM J. Computing,* vol. 9, pp. 809-826, 1980.
[8] M. Ehrgott, *Multicriteria Optimization.* Berlin: Springer, 2000.
[9] S.K. Goel and E.J. Marinissen, "Cluster-Based Test Architecture Design for System-on-Chip," *Proc. VLSI Test Symp.*, pp. 259-264, 2002.
[10] P. Harrod, "Testing Re-Usable IP: A Case Study," *Proc. Int'l Test Conf.*, pp. 493-498, 1999.
[11] Y. Huang et al., "On Concurrent Test of Core-Based SOC Design," *J. Electronic Testing: Theory and Applications,* vol. 18, pp. 401-414, Aug. 2002.
[12] V. Iyengar and K. Chakrabarty, "System-on-a-Chip Test Scheduling with Precedence Relationships, Preemption, and Power Constraints," *IEEE Trans. Computer-Aided Design,* vol. 21, pp. 1088-1094, Sept. 2002.
[13] V. Iyengar, K. Chakrabarty, and E.J. Marinissen, "Test Wrapper and Test Access Mechanism Co-Optimization for System-on-Chip," *J. Electronic Testing: Theory and Applications,* vol. 18, pp. 213-230, Apr. 2002.
[14] V. Iyengar, K. Chakrabarty, and E.J. Marinissen, "Efficient Test Access Mechanism Optimization for System-on-Chip," *IEEE Trans. Computer-Aided Design,* vol. 22 May 2003.
[15] W. Jiang and B. Vinnakota, "Defect-Oriented Test Scheduling," *Proc. VLSI Test Symp.*, pp. 433-438, 1999.
[16] S. Koranne, "On Test Scheduling for Core-Based SOCs," *Proc. Int'l Conf. VLSI Design,* pp. 505-510, 2002.
[17] E. Larsson and Z. Peng, "Test Scheduling and Scan-Chain Division under Power Constraint," *Proc. Asian Test Symp.*, pp. 259-264, 2001.
[18] E.J. Marinissen et al., "A Structured and Scalable Mechanism for Test Access to Embedded Reusable Cores," *Proc. Int'l Test Conf.*, pp. 284-293, 1998.
[19] E.J. Marinissen, S.K. Goel, and M. Lousberg, "Wrapper Design for Embedded Core Test," *Proc. Int'l Test Conf.*, pp. 911-920, 2000.
[20] E.J. Marinissen, "The Role of Test Protocols in Automated Test Generation for Embedded-Core-Based System ICs," *J. Electronic Testing: Theory and Applications,* vol. 18, pp. 435-454, Aug. 2002.
[21] E.J. Marinissen, V. Iyengar, and K. Chakrabarty, "A Set of Benchmarks for Modular Testing of SOCs," *Proc. Int'l Test Conf.*, pp. 519-528, 2002. (Benchmarks available at http://www.extra.research.philips.com/itc02socbenchm).
[22] M. Nourani and C. Papachristou, "An ILP Formulation to Optimize Test Access Mechanism in System-on-Chip Testing," *Proc. Int'l Test Conf.*, pp. 902-910, 2000.
[23] N.A. Touba and B. Pouya, "Using Partial Isolation Rings to Test Core-Based Designs," *IEEE Design and Test of Computers,* vol. 14, pp. 52-59, Oct.-Dec. 1997.
[24] P. Varma and S. Bhatia, "A Structured Test Re-Use Methodology for Core-Based System Chips," *Proc. Int'l Test Conf.*, pp. 294-302, 1998.
[25] D. Zhao and S. Upadhyaya, "Adaptive Test Scheduling in SoC's by Dynamic Partitioning," *Proc. Int'l Symp. Defect and Fault Tolerance in VLSI Systems,* pp. 334-342, 2002.
[26] Y. Zorian, E.J. Marinissen, and S. Dey, "Testing Embedded-Core-Based System Chips," *Computer,* vol. 32, no. 6, pp. 52-60, June 1999.

Vikram Iyengar received the BE degree in electrical engineering from the Birla Institute of Technology, India, in 1996, the MS degree in computer enginering from Boston University in 1998, and the PhD degree in computer engineering from Duke University in 2002. He is now an advisory engineer with the ASICs Test Methodology Group at IBM Microelectronics, Essex Junction, Vermont. He is a recipient of the Boston University ECE Department Chair Fellowship award and the IBM Graduate Fellowship award. He is coauthor of *Test Resource Partitioning for System-on-a-Chip* (Kluwer, 2002). His research interests lie in system-on-chip design and test. He has published more than 30 journal articles and refereed conference papers in the area of VLSI design and test. He is a member of ACM Sigda and Sigma Xi and a student member of the IEEE.

Krishnendu Chakrabarty received the BTech degree from the Indian Institute of Technology, Kharagpur, in 1990, and the MSE and PhD degrees from the University of Michigan, Ann Arbor, in 1992 and 1995, respectively, all in computer science and engineering. He is now an associate professor of electrical and computer engineering at Duke University. During 2000-2002, he was also a Mercator Visiting Professor at the University of Potsdam in Germany. He is a recipient of the US National Science Foundation (NSF) Early Faculty (CAREER) award, the US Office of Naval Research (ONR) Young Investigator award, and the Humboldt Research Fellowship from the Alexander von Humboldt Foundation. His current research projects (supported by NSF, US Defense Advanced Research Projects Agency, ONR, US Army Research Office, and industrial sponsors) are in system-on-a-chip test, embedded real-time systems, distributed sensor networks, and modeling, simulation, and optimization of microelectrofluidic systems. He is a coauthor of two books: *Microelectrofluidic Systems: Modeling and Simulation* (CRC Press, 2002) and *Test Resource Partitioning for System-on-a-Chip* (Kluwer, 2002), and the editor of *SOC (System-on-a-Chip) Testing for Plug and Play Test Automation* (Kluwer 2002). He has published more than 130 papers in archival journals and refereed conference proceedings, and he holds a US patent in built-in self-test. He is a recipient of a best paper award at the 2001 Design, Automation, and Test in Europe (DATE) Conference. Dr. Chakrabarty is an associate editor of the *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* and *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, and an editor of the *Journal of Electronic Testing: Theory and Applications (JETTA)*. He was the guest editor of a special issue of *JETTA* on system-on-a-chip testing published in August 2002. He was also a guest editor in 2001 of a special issue of the *Journal of the Franklin Institute* on distributed sensor networks. He is a senior member of the IEEE and a member of the IEEE Computer Society, ACM, and ACM SIGDA, and Sigma Xi. He serves as vice chair of Technical Activities on the IEEE's Test Technology Technical Council and is a member of the program committees of several IEEE/ACM conferences and workshops.

Erik Jan Marinissen received the MSc degree in computing science in 1990 and the Master of Technological Design (MTD) degree in software technology in 1992, both from Eindhoven University of Technology in Eindhoven, The Netherlands. Currently, he is a principal scientist at Philips Research Laboratories in Eindhoven, The Netherlands. His research interests include all topics in the domain of test and debug of digital VLSI circuits. He has published more than 70 journal and conference papers, holds two US patents, and has several US and EP patents pending in the domain of core test and other digital test fields. He is recipient of the Best Paper Award of the Chrysler-Delco-Ford Automotive Electronics Reliability Workshop 1995 and the IEEE Board Test Workshop 2002. He is a senior member of the IEEE and a member of the IEEE Computer Society, VIE, XOOTIC, and Philips CTAG. He served as editor-in-chief of the IEEE P1500 Standard for Embedded Core Test and is a member of the organizing and program committees of DAC, DATE, DDECS, ETW, Euro DesignCon, ITSW, LATW, SBCCI, TECS, and VLSI-SOC. He has presented numerous tutorials on core-based testing within Philips and at international conferences. He serves as a member of the editorial board of the *Journal of Electronic Testing: Theory and Applications*.