Solving the SoC Test Scheduling Problem Using Network Flow and Reconfigurable Wrappers

Sandeep Koranne
Tanner Research Inc.
sandeep.koranne@tanner.com

Abstract

Test scheduling for core-based SoCs is a challenging problem. Test schedules must be crafted with the objectives of minimizing testing time and ATE vector memory requirements, to reduce test cost, under the constraints of total available test access mechanism (TAM) width. Prior research in test scheduling has mainly used search procedures like ILP and rectangle packing to solve this problem, but these approaches are inherently computationally expensive. In this paper we describe a novel algorithm to solve the test scheduling problem using a combination of network flow algorithms, malleable job scheduling and reconfigurable wrapper design. Our approximation algorithm has polynomial time complexity and produces schedules close to the theoretical lower bound. Extensive experimental results using the new ITC'02 SoC benchmarks validate the quality of our solutions.

1 Introduction

The widening gap between VLSI system capacities and design engineering capability, in a limited time to market scenario, has prompted many design groups to adopt a policy of design reuse at the core level [16]. Typical cores include CPUs like MIPS and ARM, network controllers, embedded memories, DSP cores and associated peripherals like the IEEE 1394 Firewire and UARTs. As has been noted in [28], reusability of design alone is not sufficient as the verification and test generation efforts now dominate the typical design time. Reusability of tests is crucial for reducing total design time. This raises the problem of test knowledge transfer and physical test application. The proposed IEEE P1500 (SECT) [8] standard provides facilities for test knowledge transfer using Core Test Language (CTL) [15] and has advocated the use of a core test wrapper to facilitate modular test of embedded cores. The complete problem of core test application can be divided into three sub-problems:

1. **Top level TAM partitioning**: As the number of test pins available at the IC level is constrained, an optimal partition of these test bits must be done so as to reduce the total test cost. Test cost in this paper refers to the test application time on the automatic test equipment (ATE). The test application model assumed in this paper is based on a TESTBUS model which is a special case of the model proposed in [21]. In [21], Marinissen *et al.* propose a structured and scalable method of test access to embedded cores. Their design separates the problem of providing test access into (a) test data transport, accomplished using a test access mechanism (TAM) called a TestRail, and (b) test application to cores, accomplished using a wrapper called a TestShell. A similar approach is advocated in the upcoming IEEE P1500 standard,

2. **Test port partitioning at core level**: The second problem is scan chain (or test ports) partitioning at the core level; given a particular width of the TAM, how should the core level scan chains be connected so as to reduce the length of the longest scan chain for that core,

3. **Test scheduling**: Given a set of tests and the test resources like TAMs (many of which are shared between many cores), determine the TAM assignment to cores during the test schedule, such that the total SoC testing time is reduced.

These problems are referred to in literature as (i) the TAM design problem [3], (ii) the wrapper design problem [22] and (iii) the test scheduling problem [2], respectively. In this paper we focus our attention on the test scheduling problem.

Notation: Throughout this paper we refer to the top-level TAM width by W, the number of TAMs by B, the width of the TAM assigned to a core by w_i, and the number of cores by N_C. Each core has a test T_i. Test T_i is characterized by its number of patterns p_i and by the bitwidth[1]

[1] Bitwidth for a core is defined as the minimum TAM width beyond which there is no decrease in the test time for a core.

0-7695-1881-8/03 $17.00 © 2003 IEEE

of the core, ϕ_i. The schedule is denoted by Σ and the TAM assignment vector by π. The schedule contains tuples of the form $(T_i, w_i, t_1 \leftrightarrow t_2)$ to denote the width of the TAM responsible for executing test T_i from time instance t_1 up-to time instance t_2. The length of the schedule (makespan) is denoted $|\Sigma|$. The assignment vector π is defined as $\pi = \{w_i\} : \sum_{i=1}^{B} w_i \leq W$. The test time function is denoted $F_T(C, W)$ to represent the time for testing C on a TAM width of W. The entity C may be an individual core, in which case F_T depends only on the wrapper design, or C can be a *collection* of cores (like an SoC), in that case F_T depends on both the wrapper design and TAM design.

The remainder of this paper is organized as follows. In the next section we survey previous work in the field of TAM and wrapper design, and SoC test scheduling. In Section 3 we describe the reconfigurable wrapper design we have used; in Section 4 we show that using reconfigurable wrappers embedded core test scheduling can be represented as a malleable job scheduling problem. In Section 5 we present the network flow model of the embedded core-based SoC test scheduling problem. Experimental results on the new ITC'02 benchmarks [23] are presented in Section 6. We conclude the paper in Section 7.

2 Prior work

Prior research in TAM design has examined the use of a dedicated test bus [26], reuse of the existing system bus [6], and a scalable bus-based architecture called TESTRAIL [21]. TAM optimization for testing time minimization was investigated in [3, 13], and TAM optimization under power and routing constraints was studied in [9]. A novel re-configurable wrapper design was proposed by Koranne in [17] which allows for a dynamic change in the width of the TAM executing a core test. We shall use this wrapper design to model the core test scheduling as a malleable job rather than a static job. In addition, several techniques for SoC test scheduling, independent of TAM optimization, have been proposed in the literature. These include combinatorial optimization [25], test reordering for a large batch of ICs [14], Macro Test and test protocol scheduling [24], integer linear programming [2], and power-constrained scheduling [4, 20, 27]. All these papers have studied TAM optimization and test scheduling as separate problems. However, test access architectures and the test schedules executed on these architectures are interrelated. A test schedule that is designed for a specific TAM structure can significantly impact testing time, if applied to a different TAM design. Therefore, in order to achieve a tightly-integrated, effective SoC test architecture, it is required that TAM design and test scheduling be carried out in conjunction.

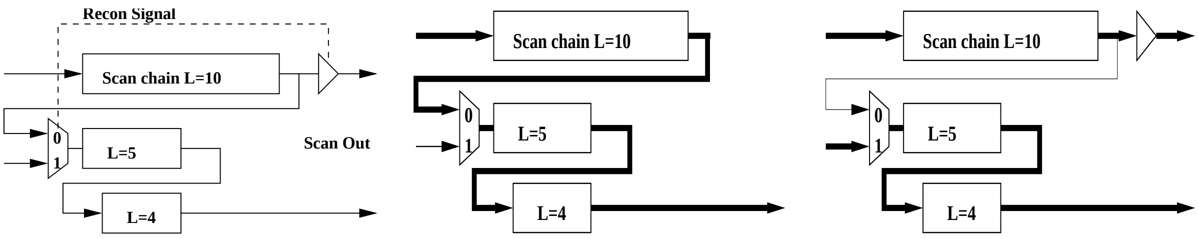
While solutions to the problem of integrated TAM design and test scheduling were proposed in [12, 18, 19], all of these methods are based on search procedures like integer linear programming, enumeration, and more recently 2-d bin packing in the form of rectangle packing. In particular, the use of rectangles for representing SoC test schedules was proposed as an attractive, intuitive interface for test integrators in [4, 7, 12, 20]. We compare the results of our proposed algorithm with exact methods based on ILP and rectangle packing methods in Section 6.

In this paper we propose a novel formulation of the SoC embedded core test scheduling (ECTSP) problem as a malleable job scheduling problem using the Reconfigurable Core Wrapper design as presented in [17]. We solve this malleable job scheduling problem using a simple reduction to network flow where the automatic test equipment is treated as a source of *test bits* and cores are modelled as *consumers* or *terminals*. The TAM responsible for transporting the test data are treated as *channels* with capacities proportional to the TAM width. We first describe the general idea of reconfigurable wrapper as presented in [17] below.

3 Reconfigurable Core Wrappers

In [17] Koranne describes a novel design of reconfigurable core wrapper which enables a dynamic change in the width of the TAM executing a core test. The central idea is to put multiplexers at the head of certain scan chains (the exact method of choosing the scan chains which can be reconfigured is described in the original paper [17] and is based on a graph theoretic representation of the scan chains). Using these multiplexers different scan chain configurations can be formed, an example is shown in Figure 1(a), (b) and (c). In Figure 1(a) the core consists of three scan chains of length 10, 5 and 4, respectively. It is obvious that an optimal partition of these scan chains for wrapper design is $\{$10,5,4$\}$ when connected to a TAM of width 1 bit, and $\{$10$\}$,$\{$5,4$\}$ when connected to a TAM of 2 bit width. Testing of this core on TAM of 3 bits or more would not reduce the test time as the length of the longest scan chain (10) cannot be decreased. Hence, $\phi = 2$ for this core. It is also clear that in any scan chain configuration the scan chain of length 10 (let us denote that by $L10$) is always connected directly to the scan input, and likewise the scan out of $L4$ is always connected to the TAM output. The rest of the scan connections depend on the TAM width. Hence, we put a multiplexer on the head of the $L5$ which is controlled by a signal $recon\_signal$. When $recon\_signal = 0$ this forms a scan connection which is optimal for TAM width of 1 bit, as shown in Figure 1(b), but if we want to test the core with 2 bits of TAM then we can set $recon\_signal = 1$, and the appropriate scan connections are formed as shown in Figure 1(c).

This idea can be generalized to more than one scan con-



(a) Multiplexers (b) TAM Width = 1 (c) TAM Width = 2

Figure 1. Reconfigurable Core Wrapper Design

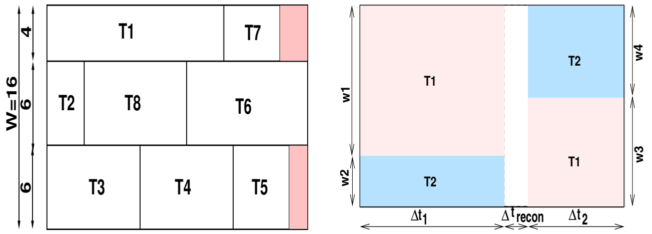
figurations, multiple scan chains, and even multiple cores in the SoC. Typically, the $recon\_signal$ will be encoded and implemented as a test control signal either from a core internal *test control block* or from a SoC wide TAP controller. Routing issues, timing issues have been dealt with and explained in [17]. Area and performance effects of the multiplexers on the scan chain have also been analyzed and found to be of little impact, as the multiplexer can be removed from the functional path.

An important point regarding the use of reconfigurable core wrappers is the fact that the scan chain reconfiguration can be changed dynamically while the core test is in progress; by halting the scan shifting and switching to a different test modes the TAM external width can be efficiently utilized. This has been used in the original paper to derive an algorithm which utilizes the idle TAM bits to optimize core test schedules. But the algorithm only optimizes idle space on TAMs at the end of a schedule. In this paper we propose a better solution by formulating core test scheduling as a malleable job and solving it using network flow.

4 Malleable Job Scheduling

Previous approaches (e.g., [3, 10]) to solve the core test scheduling problem have treated it as the minimum makespan job scheduling problem on unrelated parallel machines, denoted $R||C_{\max}$ in Operation Research terminology. A schedule obtained using the above model can be depicted as a Gantt chart as shown below in Figure 2(a). The TAM width is shown on the Y-axis and the X-axis denotes test time. The individual rectangles correspond to core tests being executed on the TAM. The width of the TAM assigned to a core does not change for the duration of the test. Thus, the jobs are treated as *static jobs*.

In the previous section we have seen that by using reconfigurable core wrappers the width of a TAM executing a core test may be changed dynamically. Thus, it is possible for core T_1 to start executing its test on a width w_1 and during the test (lets say that after p_1 patterns have been executed) the scan shifting is stopped, and the scan configuration is changed to form an optimal wrapper for TAM width w_2, and the remainder of the core patterns $p_A - p_1$ are executed on a TAM width of w_2. Such a schedule for cores T_1 and T_2 is shown in Figure 2(b).



(a) Static jobs (b) Malleable jobs

Figure 2. Job scheduling Gantt chart

In Figure 2(b) core T_1's test is executed on a width of w_1 for a period Δt_1, then there is a small reconfiguration time period, and testing of core T_1 resumes on width w_3 for time Δt_2. Cores T_1 and T_2 are equipped with reconfigurable wrappers which form balanced core scan chains at width w_1, w_3 and w_2, w_4 for core T_1 and T_2 respectively. Thus, embedded core test scheduling for cores with reconfigurable core wrappers can be modelled as malleable job scheduling problem.

An interesting observation can be made by comparing Figures 2(a) and (b); in (a) there is idle time on the 2 TAMs of width $w = 4$ and $w = 6$ (the idle time is at the end of the schedule), but the malleable job schedule by definition cannot have any idle time (even when the scheduling formulation has precedence constraints or test conflicts), as any idle time on any TAM can be re-assigned to a TAM which is currently executing a core test. Using this information a simple heuristic would be optimize static job schedules by pushing idle time on TAMs to executing TAMs as a post-processing step. This heuristic does not handle cases when the idle time is sandwiched in between core tests; hence, we propose a better model of ECTSP by formulating it as a network flow problem.

5 ECTSP as a Network Flow Problem

The automatic test equipment (ATE) can be thought of as the *source* of all the test bits needed by cores to complete their tests. The cores can be represented as *terminals* with *demands* of b_i test bits (for test stimuli). Test control bits are assumed to be handled separately using either core level test control blocks or top level test controllers. The TAMs responsible for transporting the test data to and from the cores can be modelled as *channels* with finite capacities. Obviously, the TAM which is the last to finish transporting data will constitute the schedule makespan. With this background we are now able to define the network flow model of ECTSP formally.

Let $G = (V, E)$ be a capacitated directed graph with edge capacities $c : E \rightarrow \mathbb{R}^+$, a source s and k commodities with terminals t_i and demands $d_i \in \mathbb{R}^+$, $1 \leq i \leq k$. A vertex may contain a number of terminals. For each i, we would like to route d_i units of commodity i along a path from s to the corresponding terminal so that the total flow through an edge e is at most its capacity $c(e)$. There exists a *flow*, respecting the capacities iff the *cut condition* is met:

> For any set S with $s \notin S$, the total demand of terminals within S is at most the total capacity of the edges entering S.

An excellent source of more information on the topic of network flows is the book by Ahuja et. al. [1].

Let us assume w.l.o.g. that there are N_C tests for the N_C core in a system. Each of these N_C tests has a testability requirement of b_i bits to be transported to and from the core, $1 \leq i \leq N_C$. Let there be B test resources like TAMs of width $w_1 \leq w_2 \leq \ldots \leq w_B$ bits or BIST resources.

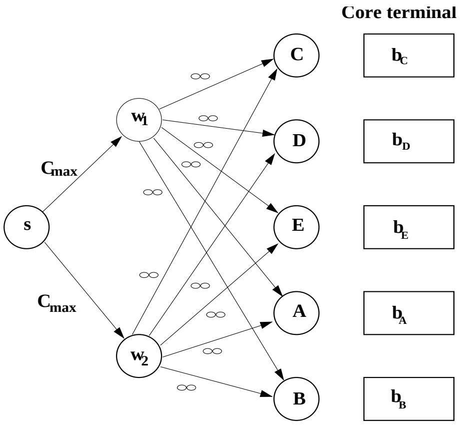
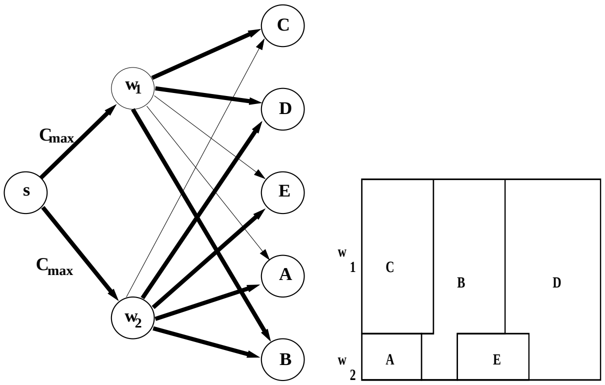


Figure 3. ECTSP as Network Flow

We now construct a graph $G = (N_C \cup B \cup s, E)$, where B is the set of test resources, N_C is the set of tests and s is a source vertex. Each test vertex i has a terminal which has an associated demand b_i of a commodity (the commodities are test bits). The edge set E is defined as follows:

We define a parameter $C_{\max}$ to be the measure of the total number of test bits that will be transported across the slowest test resource (or generated at a BIST engine). For each test $t_i \in N_C$ which can be executed with test resource w_j we add an edge between vertex i and j. The capacity of this edge e_{ij} is ∞. We add an edge e_{sj} between s and each test resource j of capacity $C_{\max}$ for TAMs. The quantity $C_{\max}$ is an estimate of the expected schedule makespan and is computed in number of bits to be transported for all the core tests. We then compute a maximum-flow of this graph; the flow can be computed in time $O(B{N_C}^2 \log N_C)$.

An illustrative example is shown in Fig. 3. If there is a flow in G such that all demands are satisfied without violating the edge capacities, then the number of bits flowing across the slowest TAM (or generated at a BIST) is at most $C_{\max}$. By dividing $C_{\max}$ by the clock frequency α, we will get a schedule of makespan at most $\frac{C_{\max}}{\alpha}$ time units. Once we have the flow for G, the test schedule can be calculated by noting that the path the commodity i takes from s to t_i is exactly the resource mapping we need. Re-ordering of tests per test resource would not change the schedule, and we can calculate the starting time for each test per test resource locally.



(a) Network flow (b) Test schedule

Figure 4. Converting flow to schedule

The general outline of the algorithm is as follows; the biggest core is scheduled on the widest TAM; if there is a *residual capacity* on that TAM then the next core is also scheduled on the same TAM. Residual capacity is initialized as the estimate of the schedule makespan. As cores are assigned to TAMs the residual capacity is reduced. Assume that core i is being considered for assignment on TAM j, but the residual capacity of TAM j does not satisfy the demand of b_i bits, then a *partial* assignment is done so that core i is assigned TAM j and TAM $j + 1$. If TAM $j + 1$ does not exist, then core i is assigned TAM j till the time the residual capacity is met, and then reconfigured to execute on all W total top level TAM bits. The flow which is returned from the above method might have test bits of core i being transported on TAMs w_1 *and* w_2. Also TAM w_2 might be serving another core also. This implies that core i needs reconfiguration. In Figure 4(a) the resultant flow for a simple SoC with two TAMs is shown. The corresponding schedule is shown in Figure 4(b). It should be noted that at most only B cores can then have reconfiguration, where B is the number of TAMs.

6 Experimental Results

We have implemented the network flow based scheduling algorithm presented in this paper (termed NFRECON). The algorithms were implemented using C++ language, and the benchmarks were executed on an Intel Pentium Celeron 1.2GHz machine with 768MB RAM running Linux. We present the results of our experiments on the ITC'02 SoC Benchmark Suite [23] in Table 1. The names of the SoCs are a measure of their test complexity. SOC d695 is an academic SoC from Duke University consisting of ISCAS benchmark circuits. The other benchmark SoCs are industrial ICs from Philips. SOC p22810 contains 6 memory cores and 22 scan-testable logic cores. SOC p34392 contains 15 memory cores and 4 scantestable logic cores. SOC p93791 contains 18 memory cores and 14 scan-testable cores.

The test planner software we have written accepts SoC data in the ITC'02 benchmark format, and proceeds to create optimized TAM architecture for a user specified top level TAM width. Then it uses this architecture to schedule each of the core test, and reports the makespan (the schedule time) in clock cycles. The software also prints useful diagnostic information and graphical plots. A number of experiments for varying W were conducted, and the results are shown below. All schedules from NFRECON were obtained within 1 second of runtime. For comparison, we have also given alongside the results given by Iyengar *et. al.* in [10]. It should be noted that the method of [10] is based on ILP (which is not scalable). When compared to the approximate methods presented by the same authors in [11], our architecture+ scheduling algorithms outperforms it as seen from the next column. We compare the schedules with rectangle packing based methods as proposed by Iyengar et. al [12] in the third column. For the comparison with test-rail based methods we have compared our results against those presented by Goel and Marinissen in [5]. One significant improvement of our method is that the width of the top level TAM can be less than the number of cores, this is a limitation of [5] and hence some of the values for small W are omitted in the experiment. For an empirical comparison we also show a lower bound for each schedule; the lower bound was computed using the relation:

$$F_T(C, W) \geq \left\lceil \frac{\sum_{i=1}^{C} F_T(i, 1)}{W} \right\rceil$$

The columns labelled ΔT show the relative comparison in percentage from the computed lower bound for each of the methods; $\Delta T = \frac{T-LB}{LB} * 100$. In the last column we present the schedule makespan obtained using our proposed method using network flow and reconfigurable wrappers (NFRECON).

7 Conclusion

In this paper we have described a method of scheduling core tests for SoCs using techniques from malleable job scheduling, network flow and reconfigurable wrapper design. We have presented a polynomial time algorithm which works very well in practice as demonstrated by our experiments on the ITC'02 benchmarks. Further work needs to be performed in integrating precedence, power and test conflict constraints into our formulation, and these are future directions of research for us.

References

[1] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows - Theory, Algorithms, and Applications*. Prentice-Hall, 1993.

[2] K. Chakrabarty. "Test Scheduling for Core-Based Systems Using Mixed-Integer Linear Programming". *IEEE Trans. on CAD*, 19(10):1163–1174, October 2000.

[3] K. Chakrabarty. "Optimal Test Access Architectures for System-on-a-Chip". *ACM Tran. Design Automation of Electronic Systems*, 6:26–49, January 2001.

[4] R. M. Chou, K. K. Saluja, and V. D. Agrawal. "Scheduling Tests for VLSI Systems Under Power Constraints". *IEEE Trans. VLSI Systems*, 5(2):175–185, 1997.

[5] S. K. Goel and E. J. Marinissen. "TAM Architectures and Their Implication on Test Application Time". In *Digest of Papers of IEEE Intl. Workshop on Testing Embedded Core-Based Systems (TECS)*, pages 3.3–1–10, Marina del Ray, CA, May 2001.

[6] P. Harrod. "Testing Reusable IP - A Case Study". In *Proc. IEEE Intl. Test Conf. (ITC)*, pages 493–498, September 1999.

[7] Y. Huang, W. T. Cheng, C. C. Tsai, N. Mukherjee, O. Samman, Y. Zaidan, and S. M. Reddy. "Resource Allocation and Test Scheduling for Concurrent Test of Core-Based SOC Design". In *Proc. Asian Test Symposium (ATS)*, pages 265–270, November 2001.

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

[9] V. Iyengar and K. Chakrabarty. "Test bus sizing for system-on-a-chip". *IEEE Trans. on Comp.*, 51:449–459, May 2002.

[10] V. Iyengar, K. Chakrabarty, and E. J. Marinissen. "Test Wrapper and Test Access Mechanism Co-Optimization for System-on-a-Chip". In *Proc. IEEE Intl. Test Conf. (ITC)*, pages 1023–1032, October 2001.

[11] V. Iyengar, K. Chakrabarty, and E. J. Marinissen. "Efficient Wrapper/TAM Co-Optimization for Large SOCs". In *Proc. Design, Automation, and Test in Europe (DATE)*, pages 491–498, March 2002.

Table 1. Comparison of schedules on published benchmark SoCs

			ILP method [10]		Par_eval [11]		Rect. pack. [12]		Cluster [5]		NFRECON	
S	W	LB	$\lvert\Sigma\rvert$	ΔT	$\lvert\Sigma\rvert$	ΔT	$\lvert\Sigma\rvert$	ΔT	$\lvert\Sigma\rvert$	ΔT	$\lvert\Sigma\rvert$	ΔT
d	16	41231	41949	1.74	42644	3.43	44545	8.03	44340	7.54	43524	5.56
6	24	27487	28327	3.05	30032	9.26	31569	14.85	30021	9.21	30097	9.50
9	32	20615	21423	3.91	22268	8.02	23306	13.05	23488	13.94	22308	8.21
5	40	16492	17210	4.35	18448	11.86	18837	14.21	19034	15.41	18079	9.62
	48	13743	16403	19.36	15300	11.32	16984	23.58	16194	17.83	16063	16.88
	56	11780	13023	10.55	12941	9.80	14974	27.11	13479	14.42	14178	20.35
	64	10307	12327	19.60	12941	25.5	11984	16.27	11033	7.04	12388	20.19
p	16	412538	462240	12.04	468011	13.45	489192	18.58	-	-	487434	18.15
2	24	275025	361576	31.47	313607	14.02	330016	19.99	-	-	328631	19.49
2	32	206269	312662	51.58	246332	19.42	312662	51.57	259975	26.03	240689	16.68
8	40	165015	278360	68.68	232049	40.62	278360	68.68	206205	24.96	215837	30.80
1	48	137512	268474	95.23	232049	68.74	268474	45.23	173705	26.31	173474	26.15
0	56	117868	266800	126.35	153990	30.64	266800	126.35	146390	24.20	131608	11.65
	64	103134	260638	152.72	153990	49.31	260638	152.72	133587	29.52	129483	25.54
p	16	936881	998733	6.60	1033210	10.28	1053491	12.44	-	-	1053730	12.47
3	24	624587	720858	15.41	882182	41.24	759427	21.58	876529	40.33	730399	16.94
4	32	544579	591027	26.16	663193	41.57	544579	0.0	585309	24.94	544579	0.0
3	40	544579	544579	0.0	544579	0.0	544579	0.0	544579	0.0	544579	0.0
9	48	544579	544579	0.0	544579	0.0	544579	0.0	544579	0.0	544579	0.0
2	56	544579	544579	0.0	544579	0.0	544579	0.0	544579	0.0	544579	0.0
	64	544579	544579	0.0	544579	0.0	544579	0.0	544579	0.0	544579	0.0
p	16	1707095	1771330	3.76	1786200	4.63	1932331	13.19	-	-	1718171	0.64
9	24	1138063	1188080	4.39	1209420	6.27	1310841	15.18	-	-	1184142	4.04
3	32	853547	898131	5.22	894342	4.78	988039	15.75	-	-	880652	3.17
7	40	682838	709316	3.87	741965	8.66	794027	16.28	816972	19.64	696856	2.05
9	48	569031	624821	9.80	599373	5.33	669196	17.60	677707	19.09	592118	4.05
1	56	487741	525880	7.81	514688	5.52	568436	16.54	542445	11.21	513102	5.12
	64	426773	478882	12.21	473997	11.06	517958	21.36	467680	9.56	459761	7.73

[12] V. Iyengar, K. Chakrabarty, and E. J. Marinissen. "On Using Rectangle Packing for SOC Wrapper/TAM Co-Optimization". In *Proc. IEEE VLSI Test Symposium (VTS)*, pages 253–258, 2002.

[13] V. Iyengar, K. Chakrabarty, and E. J. Marinissen. "Test wrapper and test access mechanism co-optimization for system-on-chip". *Journal of Electronic Testing: Theory and Applications*, 18:211–228, March 2002.

[14] W. Jiang and B. Vinnakota. "Defect-oriented test scheduling". In *Proc. IEEE VLSI Test Symposium (VTS)*, pages 433–438, April 1999.

[15] R. Kapur, M. Lousberg, T. Taylor, B. Keller, P. Reuter, and D. Kay. "CTL, the Language for Describing Core-based Test". In *Proc. IEEE Intl. Test Conf. (ITC)*, pages 131–139, October 2001.

[16] M. Keating and P. Bricaud. *Reuse Methodology Manual For System-on-Chip Designs*. Kluwer Academic Publishers, 1998.

[17] S. Koranne. "Design of Reconfigurable Core Wrappers for Embedded Core Based SOC Test". In *Proc. of ISQED*, pages 106–111, March 2002.

[18] S. Koranne. "On Test Scheduling for Core-Based SOCs". In *Proc. of VLSI Design/ASP-DAC 2002*, pages 505–510, January 2002.

[19] E. Larsson and Z. Peng. "An Integrated System-on-Chip Test Framework". In *Proc. Design, Automation, and Test in Europe (DATE)*, pages 138–144, March 2001.

[20] E. Larsson and Z. Peng. "Test scheduling and scan-chain division under power constraint". In *Proc. Asian Test Symposium (ATS)*, pages 259–264, November 2001.

[21] E. J. Marinissen, R. Arendsen, G. Bos, H. Dingemanse, M. Lousberg, and C. Wouters. "A Structured And Scalable Mechanism for Test Access to Embedded Reusable Cores". In *Proc. IEEE Intl. Test Conf. (ITC)*, pages 284–293, October 1998.

[22] E. J. Marinissen, S. K. Goel, and M. Lousberg. "Wrapper Design for Embedded Core Test". In *Proc. IEEE Intl. Test Conf. (ITC)*, pages 911–920, October 2000.

[23] E. J. Marinissen, V. Iyengar, and K. Chakrabarty. ITC '02 SOC Benchmark Website. http://www.extra.research.philips.com/itc02socbenchm/.

[24] E. J. Marinissen and M. Lousberg. "The Role of Test Protocols in Testing Embedded-Core-Based System ICs". In *Proc. IEEE European Test Workshop (ETW)*, pages 70–75, May 1999.

[25] M. Sugihara, H. Date, and H. Yasuura. "A Novel Test Methodology for Core-Based System LSIs and a Testing Time Minimization Problem". In *Proc. IEEE Intl. Test Conf. (ITC)*, pages 465–472, October 1998.

[26] P. Varma and S. Bhatia. "A Structured Test Re-Use Methodology for Core-Based System Chips". In *Proc. IEEE Intl. Test Conf. (ITC)*, pages 294–302, October 1998.

[27] Y. Zorian. "A Distributed BIST Control Scheme for Complex VLSI Devices". In *Proc. IEEE VLSI Test Symposium (VTS)*, pages 6–11, April 1993.

[28] Y. Zorian, E. J. Marinissen, and S. Dey. "Testing Embedded-Core Based System Chips". In *Proc. IEEE Intl. Test Conf. (ITC)*, pages 130–143, October 1998.