Pair Balance-Based Test Scheduling for SOCs

Yu Hu[1)], Yin-He Han[1)2)], Hua-Wei Li[1)2)], Tao Lv[1)], Xiao-Wei Li[1)2)]
[1)] *Institute of Computing Technology, CAS, Beijing, 100080, China*
[2)] *Graduate School of the CAS, Beijing, 100039, China*
{huyu, yinhes, lihuawei, lvtao, lxw}@ict.ac.cn

Abstract

Along with more pre-designed and pre-verified cores are integrated into a single chip to construct an entire system, the test application time increases significantly. This paper presents a novel test scheduling solution, unlike previous techniques that take advantage of balanced scan chains of every single core, utilizing the balance of pairwise combined cores. Experimental results for two ITC'02 SOC benchmarks show that the pair balance-based test scheduling technique achieves less test time compared to the previous approaches.

1. Introduction

The reuse of pre-designed and pre-verified intellectual property (IP) cores is emerging as a promising solution to tackle the increasing system design complexity. System designers integrate various IP cores to construct an entire system on a single chip, such a system is called system-on-a-chip (SOC). While core reuse lets system designers quickly build up an SOC, it complicates the test situation. By default, input and output terminates of cores do not directly connect to pins of SOC. To control and observe deeply embedded cores, test access mechanism (TAM) is needed [1]. In general, the architectures of TAM can be categorized in two groups: bus-based and network-based. CAS-BUS [2], Test Bus [3] and TestRail [4] are typical examples of the first group, while Network-on-Chip [5, 6, 7] belongs to the second group. In addition to accessibility, isolation is required to separate cores from its surrounding logic in certain test modes. The test wrapper is responsible for separating cores from their environments, as well as adapting the test width in case of a mismatch between core I/O width and TAM width [8]. Examples are Test Collar [3] and TestShell [4].

[*] This paper is supported by the National Natural Science Foundation of China (NSFC) under grant No.90207002 and 60242001, the Sci. & Tech. Project of Beijing under grant No.H020120120130, and the Foundation of Institute of Computing Technology, CAS, under grant No. 20036160.

As test time is the vital factor impacting the test cost, various techniques for reducing test time have been reported in the literature. In the system level, scheduling cores to test in parallel is an effective approach to shorten test application time. In case of bus-based TAM, the test scheduling problem can be formally stated as follows: Given an SOC with *N* cores and total test bus width *Wmax*, design optimal core wrapper and determine optimal assignment of cores to test buses such that the total test application time is minimized. Test schedule problem has been mapped to well-known schedule problems that are proved NP-complete, such as multiprocessor open shop scheduling [9] and 2D/3D bin packing [10, 11, 12]. Then the problem is solved with some heuristic optimization algorithm, for instance, Best Fit Decreasing (BFD) [10], Integer Linear Programming (ILP) [13], Mixed Integer Linear Programming (MILP) [14, 15], Genetic Algorithm (GA) [16], Simulated Annealing (SA) [17, 18], Evolutionary Strategy (ES) [19], and graph theory [20]. Some of these researches also take into account power dissipation, route, area overhead, modular hierarchy, and so on.

As all of the abovementioned approaches are based on the balance of every single core, the test time of each core in the SOC is determined by its longest scan chain, no matter how short the other scan chains are. For hard cores whose internal scan chains can not be partitioned, don't-care (X) bits are padded in test vectors to guarantee the scan chains to be full loaded at the same time. However, don't-care bits cost vector memory depth of testers, as well as prolong the test application time.

In this paper we will present a test scheduling approach based on the pair balance of combined cores. Our test scheduling technique balances the scan chains by concatenating scan chains of paired cores, such that the test time of the pairwise core is shorter than that of testing two cores alone.

The remainder of this paper is organized as follows. Section 2 describes the vector feature of single balance and pair balance-based test scheduling, respectively, and then gives the pair balance-based scheduling scheme. Section 3 details the heuristic scheduling algorithm. Experimental results are analyzed in Section 4. Finally, Section 5 provides some conclusion remarks and future work in this direction.

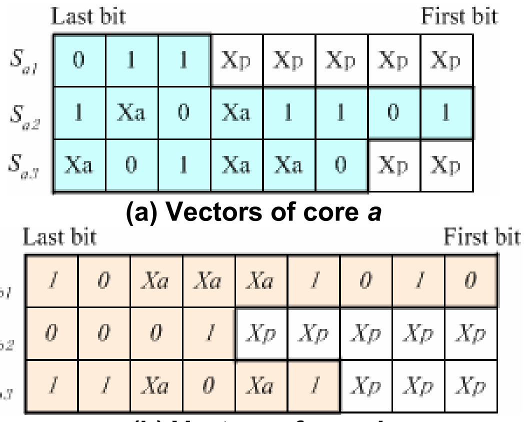
2. Vector feature and the proposed test scheme

2.1. Vector feature

2.1.1. Vectors of single balance-based test scheduling (SBTS). In general, don't-care bits in test vectors come from two ways. The one is that the automatic test pattern generation (ATPG) tools will produce some don't-care bits in the test vectors, which do not contribute to the fault coverage. This type of don't-care bits are denoted as Xa bits. The other is unbalanced scan chains, so test vectors need to be padded with don't-care bits. This type of don't-care bits are denoted as Xp bits.

Example 1: For example, Figure 1 illustrates vectors of two cores. Core *a* has three scan chains of length 3, 8 and 6 respectively, and core *b* has three scan chains of length 9, 4 and 6 respectively. Vectors of core *b* are represented in italic. Since lengths of scan chains in core *a* are unequal, vectors of S_{a1} and S_{a3} are padded to the length of vectors of S_{a2}. Hence vectors of S_{a1} have five Xp bits and vectors of S_{a3} have two Xp bits. With the same reason, vectors of S_{b2} and S_{b3} have five and three Xp bits respectively. It is evident that the better the scan chains are balanced, the less Xp bits are padded, and thus less test time is needed. Therefore, SOC integrators want scan chains as balance as possible. However, when the number of test bus wires assigned to a core is less than the number of core terminals, it will be difficult to make all of wrapped scan chains have equal length.

2.1.2. Vectors of pair balance-based test scheduling (PBTS). The PBTS technique extends the single core balance into pairwise combined balance of two cores. After replacing Xp bits in test vectors with vector bits of another core, vectors of SBTS is transformed into vectors of PBTS. Since the number of don't-care bits in vectors stored on the tester greatly decreases, the test time and vector memory depth of testers required are both significantly reduced. For example, Figure 2 shows the vectors of combined scan chains of core *a* and core *b* in Example 1. Since the shortest scan chain S_{a1} is combined with the longest scan chain S_{b2}, and the longest scan chain S_{a2} is combined with the shortest scan chain S_{b2}, the paired scan chains S_{ab1}, S_{ab2}, and S_{ab3} are balanced. Compare Figure 2 with Figure 1, 5/17*100%=29.41% of test time and vector memory depth are saved.



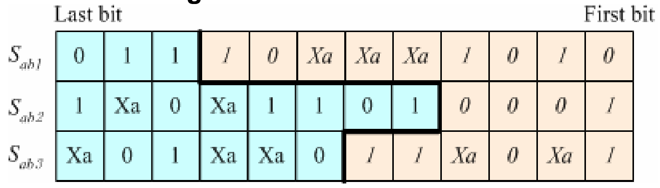
	Last bit							First bit
S_{a1}	0	1	1	Xp	Xp	Xp	Xp	Xp
S_{a2}	1	Xa	0	Xa	1	1	0	1
S_{a3}	Xa	0	1	Xa	Xa	0	Xp	Xp

(a) Vectors of core *a*

	Last bit								First bit
S_{b1}	*1*	*0*	*Xa*	*Xa*	*Xa*	*1*	*0*	*1*	*0*
S_{b2}	*0*	*0*	*0*	*1*	*Xp*	*Xp*	*Xp*	*Xp*	*Xp*
S_{b3}	*1*	*1*	*Xa*	*0*	*Xa*	*1*	*Xp*	*Xp*	*Xp*

(b) Vectors of core *b*

Figure 1. Vectors of SBTS



	Last bit											First bit
S_{ab1}	0	1	1	*1*	*0*	*Xa*	*Xa*	*Xa*	*1*	*0*	*1*	*0*
S_{ab2}	1	Xa	0	Xa	1	1	0	1	*0*	*0*	*0*	*1*
S_{ab3}	Xa	0	1	Xa	Xa	0	*1*	*1*	*Xa*	*0*	*Xa*	*1*

Figure 2. Vectors of PBTS

2.2. Test scheme of PBTS

The test architecture of PBTS is same as that of SBTS; hence there is no more hardware overhead to achieve shorter test time. However, we need to change the test scheme to benefit from the compact vectors of PBTS. Here we utilize TestRail [4] since it is flexible, scalable, compatible with IEEE P1500 [21], and widely adopted in literatures [5, 9, 22, 23].

Example 2: Figure 3 shows an example of the test architecture with three wrapped cores and four TAM wires. BR_a, BR_b and BR_c are 3-bits, 4-bits and 2-bits bypass register respectively. Core *a* has three scan chains of lengths 6, 5 and 5, core *b* has four scan chains of lengths 4, 4, 4 and 10, and core *c* has two scan chains of lengths 5 and 5. Letter "I" in blocks denotes input cells, "SC" denotes internal scan chain cells, and "O" denotes output cells. Assume core *a* is paired with core *b* to make a combined core $cocore_{ab}$, then the test stimuli of core *a* (V_a) are combined with the test stimuli of core *b* (V_b). Since core *b* is connected behind core *a*, V_b will be shifted from core *a* to Core *b* such that the scan chains of core *a* and core *b* are full loaded at the same time. After that, the test stimuli are applied to core *a* and core *b* simultaneously, and the test responses of core *a* and core *b* are captured simultaneously. For the same reason, V_a will go through the scan chains of core *b*. Without lose of the generality, assume the number of V_b is more than the number of V_a, then the test of core *a* will end earlier than the test of core *b*. Assume the remained V_b are combined with the stimuli of core *c* (V_c), since core *c* is connected behind core *b*, the combined vectors V_{bc} will first go through the bypass register BR_a of core *a*, then be shifted from core *b* to core *c*.

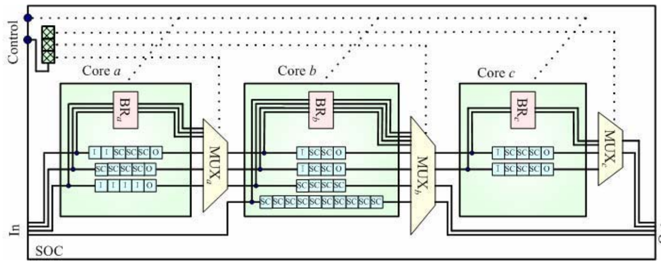


Figure 3. Test architecture

The test time of $cocore_{ab}$ is computed as follows,

$$\begin{aligned} T_{ab} = &\left(1+\max\left(\max\left(cosic\right),\max\left(soc_a\right),\max\left(soc_b\right)\right)\right) \\ &*\min\left(\#V_a,\#V_b\right) \qquad (1) \\ &+\min\left(\max\left(cosic\right),\max\left(soc_a\right),\max\left(soc_b\right)\right) \end{aligned}$$

Where *a* and *b* are cores in the SOC, $\#V_a$ and $\#V_b$ are the number of vectors V_a and V_b for core *a* and core *b* respectively; *cosic* is the length array of the ***co***mbined wrapper ***s***can-***i***n ***c***hains of core *a* and core *b*; soc_a and soc_b are the length arrays of the wrapper ***s***can-***o***ut ***c***hains of core *a* and core *b* respectively. In the case of scheduling *a* and *b* first time, $\#V_a$ and $\#V_b$ are the original number of vectors; in the case of *a* or *b* haven been scheduled previously, $\#V_a$ or $\#V_b$ is the number of vectors remained. Since scan-in and scan-out chains in equation (1) are wrapper scan-in and wrapper scan out chains, the wrapper design procedure presented in [23] is used to obtain *sic* and *soc*.

Let x_{abj} be a 0-1 variable defined as follows:
x_{abj} =1 if $cocore_{ab}$ is assigned to bus *j*
=0 otherwise
The overall test time of the SOC is:

$$T=\max_j\left\{\sum T_{ab}*x_{abj}\right\}, \quad a,b\in \mathrm{SOC}\ and\ 1\le j\le Wmax \quad (2)$$

To minimize *T*, the PBTS algorithm is proposed to find the best matched cocore, the optimal TAM assignment and the test sequence of cocores.

3. PBTS algorithm

Instead of balancing scan chains of every single core, PBTS attempts to balance combined scan chains of two cores, then assigns TAM wires and schedules cocores to minimize the test application time. Since test schedule problem is NP-complete, a heuristic algorithm PSO is applied to search the solution space to efficiently find an optimal schedule. While in this work our PBTS algorithm addresses test time optimization for hard modules with fixed-length scan chains [22], it may be easily adapted to address the test scheduling problem for soft modules with flexible-length scan chains.

During the test plan searching process, PBTS needs some searching metrics and strategy, therefore, two important definitions and one strategy are given below,

Definition (Degree of Imbalance): The Degree of Imbalance (DIB) is defined as the standard deviation of scan chain lengths. For a wrapped core, DIB is the standard deviation of wrapper scan chain lengths. For a combined core, DIB is the standard deviation of the combined wrapper scan chain lengths.

Definition (Degree of Time Approximation): The Degree of Time Approximation (DTA) is defined as the distance between two time instances, DTA=$|T_1$ -$T_2|$.

Strategy (Division and Union, D&U for short): In spite of partitioning and assigning TAM wires to cores, we need to merge the partitioned test width. Since the available test width for a core is always equal or less than the width of the free test width at any time point, TAM wires will be divided more and more slim and fragmental, thus the wrapper scan chains have to be concatenated to match the narrow test width, which results in very long test time. By setting up the upper limitation on the division index, the partitioned test buses merge once the division index is about to go beyond the upper limitation.

For the sake of the brevity, only two key functions are illustrated in the following sections. The one is the Main function, which callouts sub-functions to realize the PBTS algorithm, the other one is the FindCocore function, which searches for the optimal pairwise combined cocore with Particle swarm optimization algorithm (PSO).

3.1. Main function

The PBTS algorithm has three major steps in the Main function. Firstly, the InitialAssignment function is called to assign test buses to a cocore such that the assigned cocore has minimum DIB. Then the divison index (*di*) is calculated, which determines to use DIB or DTA metric. If *di* is less than its upper limitation (*dimax*), FindCocore is called with parameter 'BalancePair' to pair cores and select the cocore which has the minimum DIB; otherwise, FindCocore is called with parameter 'TimePair' to pair cores and select the cocore which has the minimum DTA. In FindCocore, PSO algorithm is used to find the best matched cocore. Finally, FineTune function attempts to replace the cocore occupying the bottleneck test wires to other free wires, in order to obtain shorter test time. The Pseudocode of Main function is outlined in Figure 4.

```
Main function
coreset = { core | tam_use parameter of the core equals 1, ∀core ∈ SOC }
[testplan, coreset, remwires]=InitialAssignment
If remwires>0
    While remwires>0
        Add one wire to the bottleneck cocore
        Calculate the new testtime
        If the new testtime < the original testtime
            Update testtime and testplan
        Else
            Break
        End
    End
Else
    While |coreset| > 1
        If di<dimax
            FindCocore('BalancePair')
        Else
            FindCocore('TimePair')
        End
        Delete cores in cocore from coreset
    End
End
[testplan, testtime]=FineTune
Return testplan, testtime
```

Figure 4. Pseudocode of Main function

Where *coreset* is the set of cores that use TAM wires, $|coreset|$ is the number of cores in *coreset*, *testplan* is a data structure that consists of items *cocore*, *wire*, *starttime* and *stoptime*,

$$testplan.cocore = \{cosic, core_a, core_b, sic_a, sic_b, soc_a, soc_b\}$$
$$testplan.wire = \{j \mid x_{abj} = 1, 1 \le j \le Wmax\}$$
$$testplan.starttime = the\ stoptime\ of\ the\ cocore's\ predecessor + 1$$
$$testplan.stoptime = testplan.starttime + T_{ab} - 1$$

3.2. FindCocore function

FindCocore uses PSO algorithm to find the best matched cocore. We first introduce PSO, and then explain the FindCocore function in detail.

Particle swarm optimization is an evolutionary computation technique that was first introduced by Kennedy and Eberhart [24]. PSO is motivated by the behavior of organisms such as fish schooling schooling and bird flocking, which utilizes a population-based search procedure. Comparison of stochastic local search and population based search shows that the latter performs best on problems with sharp gaps and outperformed by stochastic local search only when there are many paths to good local optima [25]. Another advantage of PSO is that very few parameters need to be adjusted to work well in a wide variety of applications.

In PSO, each individual, called particle or agent, of the population, called swarm, is initially assigned a randomized velocity and coordination, and then "flown" through the problem space. Each particle keeps track of its coordination in the problem space which is associated with the best solution (fitness) it has achieved so far. This coordination is called *pbest*. Another "best" value that is tracked by the global version of the particle swarm optimizer if the overall best value, and its coordination, obtained so far by any particle in the population. This location is called *gbest*. At each time step, each particle changes the velocity toward its *pbest* and *gbest* coordination.

The velocity and coordination are calculated with,

$$v_{id} = w*v_{id} + c_1 * r_1 * (pbest_{id} - x_{id}) + c_2 * r_2 * (gbest_d - x_{id}) \quad (3)$$
$$x_{id} = x_{id} + v_{id} \quad (4)$$

where v_{id} and x_{id} are the velocity and position of particle i in dimension d, respectively; w is the inertia weight used to balance between global and local exploration and exploitation; c_1 and c_2 are acceleration constants which represent the weighting of the stochastic acceleration terms that pull each particle toward $pbest_{id}$ and $gbest_d$ positions; r_1 and r_2 are two random numbers uniformly distributed within the range [0, 1].

The Pseudocode of FindCocore concluding PSO algorithm is as follows,

```
FindCocore function
For each particle
    Initialize $v_{id}$ and $x_{id}$
End

Do
    wc=DesignWrapper [23]
    For each particle
        Calculate fitness value p *
        If p is better than pbest
            Set p as the new pbest
    End
    Set the particle with the best fitness value as the gbest
    For each particle
        Update particle velocity according to (3)
        Update particle position according to (4)
        Decrease inertia weight w linearly
    End
While maximum iteration or minimum error is not attained
Calculate $T_{ab}$ with equation (1)
$starttime = the\ stoptime\ of\ the\ cocore's\ predecessor + 1$
$stoptime = starttime + T_{ab} - 1$
Return cocore, starttime and stoptime
```

* Depending on the parameter assigned to FindCocore, BalancePaire or TimePair is called to calculate *p*.

Figure 5. Pseudocode of FindCocore function

4. Experimental results

Experiments were performed with two ITC'02 SOC test benchmarks d281 and f2126 [26]. In this work, we do not address the hierarchical structure, the interconnect test and BIST of the benchmarks. The PBTS algorithm is implemented in Matlab version 6.1 [27], executed on a PC with Pentium 4 1.4 GHz processor and 256 MB memory.

In our experiments, the maximum number of iterations in PSO algorithm was set to 100; the desired error criteria was 0; the number of particles was 20; the inertia weight *w* was gradually decreased from 0.9 towards 0.4; $c_1=c_2=2$; and *Vmax*=0.1* the number of *remwires* in each dimension. There were seven TAM widths 16, 24, 32, 40, 48, 56 and 64. For each TAM width, thirty experiments were performed, starting with a swarm and velocities uniformly distributed within the range [1, *Wmax*] and [0, *Vmax*] respectively. The maximum degree of division was experientially set to the floor of *Wmax*/3.

Table 1 presents optimal results of different TAM width for the two SOC circuits. We compare our experimental results with the lower bound [9] and two efficient SBTS approaches TR [22] and TCG [20]. The row LB is the lower bound. The row TR and TCG is the experimental results of TR and TCG respectively; bold italic entry of PBTS lists our experimental results, ΔT shows the difference between the best PBTS solution and LB, which is $\Delta T = \left(T_{PBTS} - LB\right)/LB * 100\%$. The computation time of the best result is given in the CPU Time row. From the table, we can see that for the two benchmark circuits with all TAM widths, PBTS achieves the shortest test time compared to TR and TCG.

Table 1. Experimental results compared with LB [9], TR [22] and TCG [20]

SOC	*Wmax*	16	24	32	40	48	56	64
d281	LB	7636	5091	3926	3926	3926	3926	3926
	TR [22]	8276	5550	4163	3926	3926	3926	3926
	TCG [20]	8156	5830	4640	3926	3926	3926	3926
	PBTS	***8029***	***5550***	***4163***	***3926***	***3926***	***3926***	***3926***
	ΔT%	5.15	9.02	6.04	0	0	0	0
	CPU Time (seconds)	3.7	3.3	4.1	3.8	4.6	4.3	3.3
f2126	LB	335334	335334	335334	335334	335334	335334	335334
	TR [22]	372125	335334	335334	335334	335334	335334	335334
	TCG [20]	357089	335334	335334	335334	335334	335334	335334
	PBTS	***357089***	***335334***	***335334***	***335334***	***335334***	***335334***	***335334***
	ΔT%	6.49	0	0	0	0	0	0
	CPU Time (seconds)	7.5	9.3	7.7	10.0	7.8	9.7	8.2

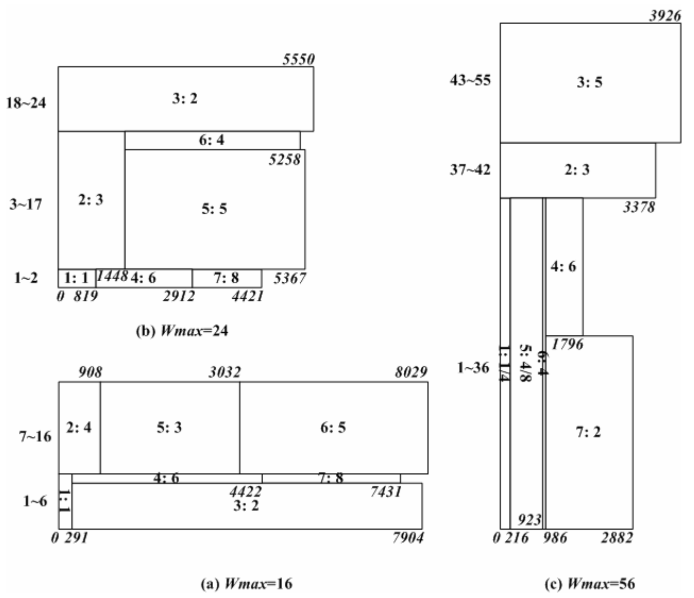


Figure 6. Test plans of d281

The test plans of d281 obtained with PBTS are shown in Figure 6. The label in each block is the sequence number of the *cocore*, a/b means the two cores that the *cocore* consists of, e.g. in (c), 5:4/8 means this is the fifth *cocore*, which is formed by core 4 and core 8. When the *cocore* has only one core, the label would be like I:a, for example, also in (a), 6:4 means the sixth *cocore* only consists of core 4. Data on the left side of the test plan indicates the TAM wires. And the test stop time of each *cocore* is represented in italic.

5. Conclusions

This paper has proposed a heuristic test scheduling technique based on the pair balance of combined cores. By replacing the useless don't care bits with vector bits of another core, the PBTS algorithm utilizes the don't-care bits padded in test vectors when TAM wires assigned to the core are less than the number of core terminals, hence reduces test application time and tester memory depth required. Moreover, while providing better test plans, PBTS adds no more hardware overhead than previous proposed single balance-based techniques. For two ITC'02 SOC benchmarks, experimental results of PBTS are compared with the theoretical lower bound on test time proposed in [9] and the experimental results of two SBTS algorithms. Comparison shows that PBTS achieves more efficient test plans than SBTS algorithms with various TAM widths. Therefore, the technique proposed in this paper outperforms other previously published single balance-based techniques by efficiently exploiting the test vector features.

As future work, we plan to extend the pair balance-based test scheduling algorithm to consider power dissipation, hierarchical structure, and interconnects.

References

[1] Y. Zorian, "Test Requirement for Embedded Core-Based Systems and IEEE P1500", *Proc. Int. Test Conf. (ITC)*, Washington DC, USA, 1997, pp. 191-199.

[2] M. Benabdenebi, 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)*, Paris, France, 2000, pp. 41-145.

[3] P. Varma and B. Bhatia, "A Structured Test Reuse Methodology for Core-Based System Chips", *Proc. Int. Test Conf. (ITC)*, Washington DC, USA, 1998, pp. 294 - 302.

[4] E.J. Marinissen, R. Arendsen, and G. Bos, et al., "A Structured and Scalable Mechanism for Test Access to Embedded Reusable Cores", *Proc. Int. Test Conf. (ITC)*, Washington DC, USA, 1998, pp. 284 - 293.

[5] P. Guerrier and A. Greiner, "A Generic Architecture for on-chip Packet-Switched Interconnections", *Proc. Design, Automation and Test in Europe (DATE)*, Paris, 2000, pp. 250 - 256.

[6] L. Benini and G. De Micheli, "Networks on Chips: A New SoC Paradigm", *Computer*, Jan. 2002, vol. 35, no. 1, pp. 70 - 78.

[7] E. Cota, M. Kreutz, and C.A. Zeferino, et al., "The Impact of NoC Reuse on the Testing of Core-Based Systems", *Proc. VLSI Test Symp. (VTS)*, California, USA, 2003, pp. 128 - 133.

[8] E.J. Marinissen, S.K. Goel, and M. Lousberg, "Wrapper Design for Embedded Core Test", *Proc. Int. Test Conf. (ITC)*, Atlantic City, NJ, USA, 2000, pp. 911-920.

[9] K. Chakrabarty, "Optimal Test Access Architectures for System-on-a-Chip", *ACM Trans. Design Automation of Electronic Systems*, Jan. 2001, vol. 6, pp. 26-49.

[10] Y. Huang, W.T. Cheng, and C.C. Tsai, et al., "Resource Allocation and Test Scheduling for Concurrent Test of Core-Based SOC Design", *Proc. Asian Test Symposium (ATS)*, Kyoto, Japan, 2001, pp. 265–270.

[11] V. Iyengar, K. Chakrabarty, and E.J. Marinissen, "On Using Rectangle Packing for SOC Wrapper/TAM Co-optimization", *Proc. VLSI Test Symp. (VTS)*, Monterey, CA, USA, 2002, pp. 253 - 258.

[12] Y. Huang, S. M. Reddy, and W.T. Cheng, et al, "Optimal Core Wrapper Width Selection and SOC Test Scheduling Based on 3-D Bin Packing Algorithm", *Proc. Int. Test Conf. (ITC)*, Baltimore, MD, USA , 2001, pp. 74-82.

[13] K. Chakrabarty, "Design of System-on-a-Chip Test Access Architecture under Place-and-Route and Power Constraints", *Proc. IEEE/ACM Design Automation Conf. (DAC)*, Bergen, Norway, 2000, pp. 432–437.

[14] K. Chakrabarty, "Test Scheduling for Core-Based Systems Using Mixed-Integer Linear Programming", *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, 2000, vol. 19, no. 10, pp. 1163–1174.

[15] V. Iyengar and K. Chakrabarty, "Precedence-Based, Preemptive, and Power-Constrained Test Scheduling for System-on-a-Chip", *Proc. VLSI Test Symp. (VTS)*, Marina Del Rey, CA, USA, 2001, pp. 368–374.

[16] S. Chattopadhyay, K.S. Reddy, "Genetic Algorithm Based Test Scheduling and Test Access Mechanism Design for System-on-Chips", *Proc. Int. Conf. VLSI Design (VLSID)*, New Delhi, India, 2003, pp. 341 -346.

[17] E. Larsson, Z. Peng, and G. Carlsson, "The Design and Optimization of SOC Test Solutions", *Proc. Int. Conf. Computer Aided Design (ICCAD)*, San Jose, CA, USA, 2001, pp. 523 - 530.

[18] Wei Zou, S.M. Reddy, and I. Pomeranz, et al., "SOC Test Scheduling Using Simulated Annealing", *Proc. VLSI Test Symp. (VTS)*, Napa Valley, CA, USA, 2003, pp. 325 - 330.

[19] Yu Xia, M. Chrzanowska-Jeske, and Benyi Wang, et al., "Using a Distributed Rectangle Bin-Packing Approach for Core-Based SOC Test Scheduling with Power Constraints", *Proc. Int. Conf. Computer Aided Design (ICCAD)*, San Jose, CA, USA, 2003, pp. 100 - 105.

[20] C. P. Su, C. W. Wu, "A Graph-Based Approach to Power-Constrained SOC Test Scheduling", *Journal of Electronic Testing: Theory and Applications*, 2004, vol. 20, no. 1, pp. 45-60.

[21] IEEE P1500 Standard for Embedded Core Test. http://grouper.ieee.org/groups/1500/

[22] S.K. Goel and E.J. Marinissen, "SOC Test Architecture Design for Efficient Utilization of Test Bandwidth", *ACM Trans. Design Automation of Electronic Systems*, October 2003, vol. 8, no. 4, pp. 399–429.

[23] 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 Application (JETTA)*, March 2002, vol. 18, no. 2, pp. 211–228.

[24] J. Kennedy and R. Eberhart, "Particle Swarm Optimization", *Proc. Int. Conf. Neural Networks*, Perth, WA Australia, 1995, vol.4, pp. 1942 - 1948.

[25] H. Muhlenbein and T. Mahnig, "A Comparison of Stochastic Local Search and Population Based Search", *Proc. Congress on Evolutionary Computation (CEC)*, Honolulu, USA, 2002, vol. 1, pp. 255-260.

[26] E.J. Marinissen, V. Iyengar, and K. Chakrabarty, "A Set of Benchmarks for Modular Testing of SOCs", *Proc. Int. Test Conf. (ITC)*, Baltimore, MD, USA, 2002, pp. 519–528.

[27] Mathworks. http://www.mathworks.com/