Using a Distributed Rectangle Bin-Packing Approach for Core-based SoC Test Scheduling with Power Constraints

Yu Xia, Malgorzata Chrzanowska-Jeske, Benyi Wang and Marcin Jeske
Electrical and Computer Engineering Department
Portland State University, Portland, OR

Abstract

We present a new algorithm to co-optimize test scheduling and wrapper design under power constraints for core-based SoCs (System on Chip). Core testing solutions are generated as a set of wrapper designs, each represented by a rectangle with width equal to the test time and height equal to the number of TAM (Test Access Mechanism) wires used. The test-scheduling problem with power constraints is formulated as the distributed rectangle bin-packing problem, which allows wrapper pins to be assigned to non-consecutive SoC pins. The generalized problem for multiple-TAMs is solved by global optimization using evolutionary strategy and the sequence-pair representation. Experiments on ITC'02 benchmarks are very encouraging.

1. Introduction

Embedded cores have become more common in large SoC-designs. They are usually deeply embedded in the system chip and direct access is often impossible. Cores have to be tested on a system level after manufacturing and special test access mechanisms (TAMs) are needed. Selecting and scheduling test solutions for SoC embedded IP cores is a very complex problem. In order to facilitate reuse of test vectors provided by the core vendor, an embedded core must be isolated from the surrounding logic and test access must be provided from the I/O pins of the SOC. A test wrapper forms the interface between the TAM and a core, while the TAM transports test data between SoC pins and the wrapper. The general problem of SoC test integration includes the design of TAM architectures, optimization of the core wrappers, test scheduling, and wrapper pin assignments. The goal is to minimize testing time given TAM architecture and power constraints.

A number of recent papers cover various aspects of SoC test scheduling. Most earlier papers propose methods to solve wrapper design and test scheduling as separate problems. Recently, [2] and [5] presented an integer-linear programming formulation of co-optimization of wrapper design and test scheduling for SoCs. In [8], modeling of the concurrent test-scheduling problem as a pseudo-3D bin-packing problem was proposed along with a best-fit heuristic method to solve it. The test scheduling problem with power constraints was considered in [3] and [8]. ILP formulation for TAM design under Place-and-Route and power constraints was presented in [9], but with only one wrapper design per core. The Multiple-TAM problem (optimal assignment of TAM wires to different partitions) was discussed in [5] and solved by enumeration of possible solutions. Problem description and conditions for using non-consecutive TAM pins was presented in [1], and later in [14], but none of the previous approaches introduced an algorithm that allows for wrapper-pin assignment to non-consecutive SOC pins.

In this paper, we addressed the SOC test scheduling and wrapper design co-optimization problem with wrapper pin assignment to non-consecutive SOC pins under power constraint. We also consider partitioning of TAM width (wires) to a given number of partitions and assignment of cores to multiple TAMs.

Given the test-set parameters for the SOC cores, the number of SOC pins (or the number of TAM wires), and the maximum power dissipation allowed during the test, our method determines an optimal test schedule, assignment of TAM wires among cores, and an optimal wrapper design for each core, such that the overall system testing time is minimized and test power dissipation is below the max dissipation limit. We also generalize this problem for multiple-TAMs. Given a number of TAMs, we globally optimize TAM width and the assignment of cores to TAMs such that the total test time is minimized.

We represent wrapper design as a rectangle with width being test time and height being the number of TAM wires. With this representation the test-scheduling problem is similar to the fixed-height floorplanning problem but with added complexity. In floorplanning, we assume the areas of modules are fixed and that aspect ratios change continuously in given ranges, while in test scheduling, areas and aspect ratios change in a non-continuous (discrete) fashion. The power dissipated during the testing of a core is associated with each rectangle. An Evolutionary Algorithm (EA) and sequence-pair (SP) representation [10] were used in our approach.

A heuristic approach using the sequence-pair representation for test scheduling problem was considered in [7], and a Simulated Annealing (SA) algorithm using the sequence-pair representation has been recently proposed by Zou et al. [14]. No power constraints were considered, and despite using a heuristic to generate an initial solution for SA solver, the algorithm in [14] is relatively slow. It will be shown in the result section that our results are much better than those in [7] and that our test time results are comparable with [14], while our CPU time is much shorter.

The remainder of this paper is organized as follows. In Section 2, we describe our overall approach. Wrapper design optimization is presented in Section 3. We discuss the test scheduling algorithm and non-consecutive pin assignment problem in Section 4. The sequence-pair representation is presented in Section 5 and our Evolutionary Algorithm in Section 6. Power constraints and multiple TAM approach are presented in section 7. Results and conclusions are given in Section 8 and Section 9, respectively.

2. Problem Formulation

Let the SoC design consist of N cores, and each core C_i, $1<=i<=N$ has n_i data inputs, m_i data outputs, b_i bidirectional data I/O, sin_i scan inputs, and $sout_i$ scan outputs. Let K be the total width of the TAMs and B be the number of TAM partitions. Also assume that each core must be tested with P_i patterns and that the maximum peak power during testing for each core is given. The amount of power dissipated during core testing depends on core switching activity and the number of FFs in a core. Power estimation is a quite challenging problem and is not a subject of this paper.

The overall problem that we solve is as follows:
Given a set of N cores, their specific test parameters, the number of SoC pins, the maximum-allowable peak power dissipation Q, and power dissipation data for each core, we design the test schedule

with TAM architecture and wrapper designs for all wrapper-based cores such that the SoC test time is minimized and the peak power during testing never exceeds Q.

There are three steps. First we generate all possible optimized wrapper designs for each core under the specified TAM width. In the next step we solve the test-scheduling problem under the maximum TAM width and power constraint using the sets of pre-designed/optimized wrapper solutions. Finally, we assign wrapper pins to non-consecutive SOC pins according to an optimized test schedule.

We generalize the same problem for multiple TAMs with a given number of TAM wires and a given number of partitions. We solve for the minimized test schedule with the optimized distribution of TAM wires among the given number of partitions.

3. Core Wrapper Design

A test wrapper is a layer of DFT logic that connects a TAM to a core for testing purposes [4]. Since large cores typically have hundreds of core terminals, and the total number of TAM wires available depends on the limited number of SoC pins [2], in practice, wrappers may often need to perform test width adoption when the TAM width is not equal to the number of core terminals. Core-internal testing is only considered in this paper. To calculate the test time, T, for a wrapper assuming different assigned TAM widths we use the well-known expression [4] given below.

$$T = \{1 + \max(S_i, S_o)\} \bullet P + \min(S_i, S_o) \qquad (1)$$

S_i is the length of a wrapper input scan chain, S_o the length of a wrapper output scan chain, and P is the number of test patterns.

For cores with internal scan chains we used an algorithm based on Best Fit Decreasing (BFD) heuristic [2]. The functional inputs, outputs and bidirectional I/O are assigned to wrapper scan chains using the same algorithm. For cores without internal scan chains we use unbalanced design [14] and allow different numbers of SoC pins to be assigned to scan in and to scan out.

As a result of the above given algorithms, each core C_i is described with a set of L_i wrapper configurations, $Z_{i1}, Z_{i2}, \ldots Z_{ij}, \ldots Z_{iLi}$ represented by a pair of numbers (W_{iZj}, $T(W_{iZj})$). W_{iZj} is the number of TAM wires of the *j-th* wrapper configuration for core i, and $T(W_{iZj})$ is the associated testing time. Similarly to previous approaches, we model the wrapper design as a rectangle with the width equal to $T(W_{iZj})$, and the height equal to W_{iZj}. As discussed in [1], for a given core, the testing time varies with TAM widths as a "staircase" function. The designs that represent the smallest TAM width for a given test time are known as Pareto-optimal designs and were formally defined in [1]. Only rectangles corresponding to Pareto-optimal TAM width need to be considered.

4. Test Scheduling problem

Given is a SOC with K TAM wires, and a set of N cores. Each core C_i $1 <= i <= N$ is represented by a set of L_i wrapper configurations. Find the assignment of core wrapper pins to the pins of SoC and determine the test starting time for each core such that the overall test time is minimized. Wrapper configurations for each core C_i are represented with a set of rectangles and for each core we want to choose one rectangle such that when packed in a bin the width of the bin is minimized and the height is no larger than K.

The classical floorplanning problem of placing a set of modules (rectangles) on a plane such that the overall area is minimized and no rectangle overlaps can be modified to represent a SoC test-scheduling problem [5, 8]. Floorplan height represents a fixed number of available TAM wires, K, and the width represent the SoC test time that has to be minimized.

A given floorplan is feasible if no rectangles overlap and the aspect ratio of the floorplan is within specified limits. No rectangle can be partitioned. However, the test schedule of a SoC is feasible if no two cores are assigned to the same SoC pin for the same testing time instance and for each core all its wrapper pins are assigned to SoC pins for the entire time needed to test that core. The height of the bin represents the TAM's pins and the order of the pins is given. Representing a wrapper design for a core as a rectangle limits the pin assignment to the consecutive pins. If we allow the rectangle to be partitioned into smaller rectangles of the same width (width represents core testing time) we relax the consecutive pin limitation. The classical floorplanning feasibility requirement restricts the wrapper pin assignment to only consecutive SoC pins.

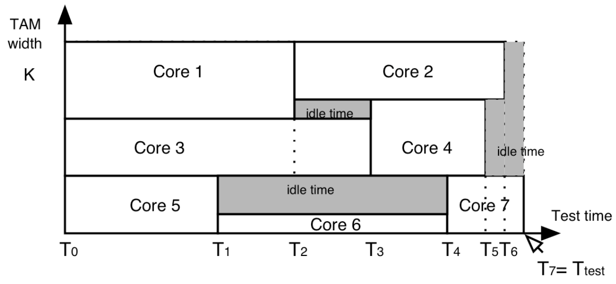

Figure 1. An example of a test-schedule floorplan

Any test schedule can be represented as a floorplan of rectangles; therefore, various representations that have been developed to represent floorplans of VLSI designs can be used for the test-scheduling problem. In recent years many new representations for floorplanning have been proposed, and a sequence pair (SP) representation [10] used in this algorithm will be discussed in the next section.

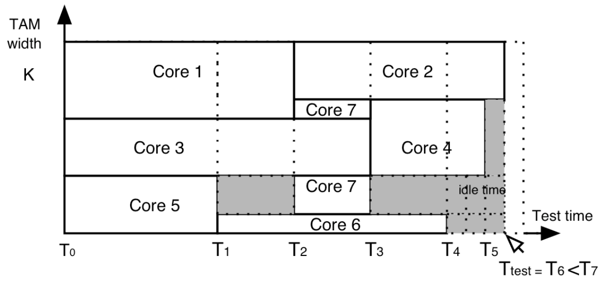

Figure 2. Test-schedule floorplan with partitioned Core 7.

An example of a test scheduling solution, represented as 2D bin packing, is shown in Fig. 1. Notice *Core 7* could be assigned to non-consecutive SOC pins and we could reduce testing time from T_5 to T_6. It would require partitioning the rectangle representing the wrapper design for *Core 7* into two rectangles of the same width (the same testing time) and heights, 1/3 and 2/3 of the original height, as shown in Fig. 2. Predicting the optimal partition for a rectangle is close to impossible, therefore we established conditions for a feasible time schedule represented with a non-feasible floorplan (shown in Fig. 3). To generate a feasible test schedule assuming that a core can use non-consecutive SOC pins we need to satisfy the following constraint.

Constraint 1. The total TAM width calculated as a sum of TAM wires used by all cores tested at a given time is smaller than K for $0 <= t <= T_{test}$ where T_{test} is the time needed to complete testing of the given SoC.

Constraint 1 can be satisfied by assuring that the total TAM width is smaller than K at each time a new core is added to the set of cores being tested. Therefore, we generate test-schedule floorplans and calculate the total TAM width at T_0 (test starting time) and at each time instance ($T_1, T_2 \ldots$) when a new core is added to the testing set. The TAM-width histogram representing the total TAM width versus test time, for the example from Fig. 2 and Fig. 3, is shown in Fig. 4.

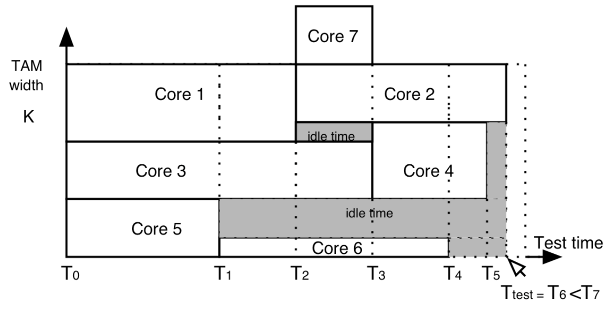

Figure 3. Test-schedule floorplan

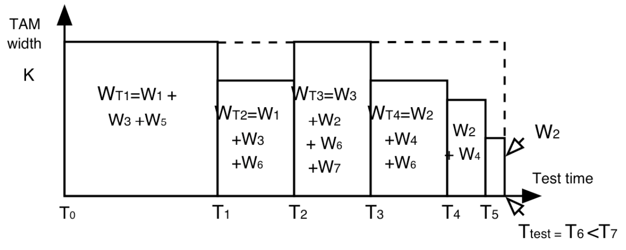

Figure 4. TAM-width histogram.

5. Sequence Pair Representation

Sequence pair representation was introduced by Murata et al. [10] to represent a non-slicing floorplans of VLSI designs. The main idea of the sequence pair representation is that two permutations (□$^+$, □$^-$) of a set of modules are sufficient to describe a packing. Each permutation imposes a specific placement for each module and there is a one-to-one relationship between a sequence pair and its packing. Specifically,

(… a … b , … a … b …) place a to the left of b

(… a … b , … b … a …) place a below b

Given a sequence pair, one can construct an oblique lattice structure with the lines labeled in the same order as they appear in (□$^+$, □$^-$). Each module is placed at the lattice point that is the intersection of the two lines with the same label as shown in Fig. 5.

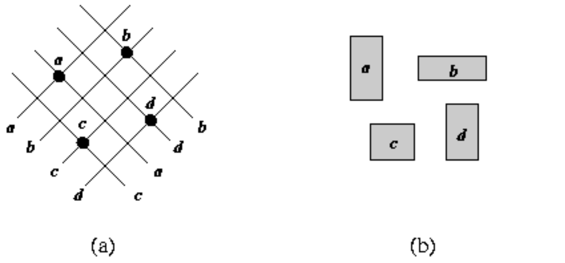

Figure 5. (a)oblique grid for □+ = {a b c d} and □- = {c a b d}, and (b) resultant placement.

Two weighted, acyclic digraphs can be constructed in which the modules at the lattice points form the vertex set. In the horizontal graph G_h an edge is placed from module i to module j iff i "is to the left of" j. Similarly, in the vertical graph G_v an edge is placed from module i to module j iff i "is below" j. Source and sink vertices are added to each graph. Fig. 6 shows this construction.

A weight is associated with each edge in the G_h (G_v) graph that indicates the width (height) of the respective module. A longest path algorithm conducted in both graphs will give the overall packing height and width. Paths in the vertical constraint graph represent the time instances at which the total number of TAM wires has to be calculated to generate the TAM-width histogram, The total TAM width is given by the length of the path.

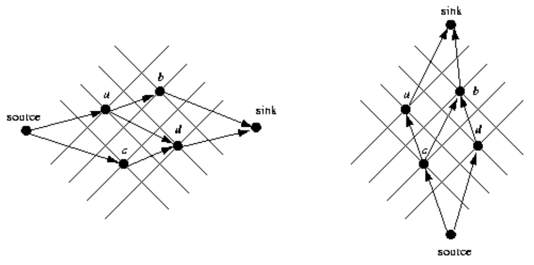

Figure 6: Weighted graphs G_h(left) and G_v(right) of the sequence pair given in Figure 5.

In our approach we use a very fast ($O(nlogn)$) algorithm [11] to translate the sequence-pair representation to its corresponding block placement. Positions of modules on a floorplan are directly generated from the sequence pair without need for constraint graphs. The algorithm is based on computing the longest common subsequence in a pair of weighted sequences. A lower complexity ($O(nloglogn)$) version of this algorithm, also proposed in [11], gives worst experimental results, perhaps due to its complex implementation.

6. Evolutionary Algorithm

Unlike *simulated annealing* (SA)□ frequently used in VLSI layout synthesis□ an *evolutionary algorithm* (EA) processes a population of potential solutions in parallel rather than a single solution. Our algorithm is derived from an *evolution strategy* (ES). Parents are subjected to stochastic "reproduction" operators that produce offsprings. There are two basic types of reproduction operators used in EAs. Recombination uses component parts from two parents to create an offspring, and mutation alters parts in a single parent to create an offspring. We only use mutation operators for reproduction as they always produce a feasible offspring from a feasible parent. Recombination was not used because it has a high probability of producing invalid structures.

Our EA begins with an initial population of $k=20$ randomly generated "parents", where a parent is encoded as a sequence pair. This population size provides a reasonable tradeoff between computation time and exploration of the solution space. Each parent undergoes reproduction to produce one offspring.

In the simple approach, parents and offspring compete equally for survival; they are ranked according to fitness and the most fit become parents in the next generation. In our algorithm we use a tournament selection process when evaluating which solutions will survive to the next generations. Instead of choosing the best k solutions based on the value of the cost function (fitness), we evaluate each solution against a randomly chosen set of TS solutions, where TS is called a tournament set size, and record the number of wins for each solution. The best scoring k solutions are chosen for the next generation. The tournament selection process allows for diversification of the solutions and so decreases search time. If properly designed, each generation will contain a subset of solutions with the same value of the cost function but with different values of the components of that function. That set of solutions is called a Pareto frontier. It is very important for multi-objective optimization, and it is a key reason we have chosen EA as our engine for the test-scheduling algorithm, a multi-objective optimization problem (even though in this work we only solve the single-objective sub-problem). The final solution can be chosen from among the solutions in the Pareto frontier. The existence of multiple solutions gives a designer the flexibility to choose one with the most important objective the best optimized.

Four types of mutation operators were used. Swap operator: exchange the position of two cores in only $\square^+$, only $\square^{\square}$, or both $\square^+$ and $\square^{\square}$. Inversion operator: two positions in a sequence pair are randomly chosen and the order of cores between these points is reversed. Insert operator: a randomly chosen core is inserted into another randomly chosen position in the sequence pair. Change operator: randomly choose a wrapper design for a randomly chosen core. The following operators are used in our algorithm: (a) invert $\square^+$ only, *(b)* insert $\square^{\square}$ only, (*c*) swap in $\square^+$ only, (*d*) swap the same two integers in both $\square^+$ and $\square^{\square}$, *(e)* swap in $\square^+$ only, and *(f)* change operator. These operators are applied with probability 0.2, 0.1, 0.2, 0.1, 0.1, and 0.3, respectively.

In floorplanning optimization the objective is to minimize the overall area of a floorplan. The objective in test-scheduling is to minimize the test time (width of a floorplan) under fixed TAM width (height of a floorplan). Therefore, we use test-time as the objective function (Cost_Function = T_{test}). Based on experiments, we have introduced heuristic steps into the EA framework, as for such implementation we were getting better results. When the maximum height of the test-time histogram (one is shown in Fig. 4) is 30% over the fixed TAM width, we choose the module with the largest TAM width value and we randomly choose another wrapper solution for that module. If the maximum height of the histogram is 30% below the fixed TAM width, we choose the module with the longest test time and will randomly choose another wrapper design for that core.

Since we solve the test-scheduling problem for distributed SoC pins, the actual pin assignment is performed after test starting times for all cores have been determined. It can be easily shown that a feasible solution to the pin assignment problem, given a feasible SoC test-scheduling solution, is always possible. We represent each chosen wrapper design as a set of rectangles each of height equal to one TAM wire and width equal to the testing time required for a given core. The problem of assigning wrapper-to-TAM pins can be modeled as a modified channel routing problem that can be exactly solved with the Left Edge Algorithm, as there are no vertical constraints in the pin assignment problem.

7. Power Constraints and Multiple TAMs

At any time during SoC testing, the maximum power dissipation cannot exceed a certain limit, *Q,* (SoC power budget). Otherwise, the chip may be damaged. P_i, represents the peak power dissipation for core *i*. In general the peak power dissipated during the test may be different for different wrapper designs. Based on wrapper design we can estimate that wider test data leads to shorter test time, but at the same time, switching activity of the core under test per cycle can increase and thus increase the maximum peak power. We ran our experiments for two cases; (1) the maximum peak power is constant for a given core, (2) the maximum peak power is a linear function of the width of the test data.

To evaluate power constraint we calculate the sum of the maximum peak powers for all cores tested at a given time instance. We assume the maximum peak power for a core in testing is the same over the entire time the core is tested. Therefore, we just need to calculate the sum of the maximum peak powers whenever a new core begins to be tested. Similarly to the TAM-width histogram, we generate a Power histogram. An example of such histogram for test schedules is shown in Fig. 7

To consider power constraints we added a penalty coefficient into the objective function. If power dissipation, is at any time larger than the power budget, *Q*, the penalty coefficient, $P_{coef.}$, is equal to the difference is equal to the difference between the highest value of the instance power, $P_{time\text{-}max}$ and the power budget *Q*. The cost function is given with the following formula cost = ($P_{time\text{-}max}$ – Q + 1) T_{test}.

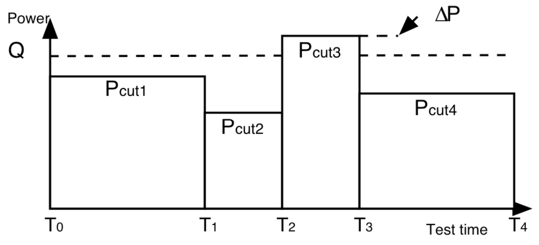

Figure 7. Power histogram

To run experiments of test scheduling under power constraints using ITC'02 benchmarks, we had to make a number of assumptions. Only one benchmark (h953) among the ITE'02 benchmark set has power dissipation numbers included. As it is specified in the ITC'02 benchmark format [13], power is expressed as a non-negative integer in the form of power dissipation per test. The unit of power is not specified, but should be the same for all modules throughout one SOC benchmark. So, we made the following assumptions. For case (1) we assume that the power numbers given in the benchmark represent the maximum peak powers dissipated during core testing, and for case (2) we assume that the given power numbers represent the maximum peak power when test is performed using only one TAM wire.

The sequence pair representation has been modified for the multi-TAM problem. We represent each TAM as a separate sequence pair, and have designed new operators for changing the number of TAM wires assigned to each partition and moving cores between different partitions.

The overall problem which we solve can be formulated as follows:

Multi-TAM Non-Consecutive Pin SoC Test Scheduling Problem under Power Constraints:

Given is a SoC with *K* TAM wires, a number of TAM partitions, *B*,, a set of *N* cores and the maximum test power budget *Q*. Each core C_i $1<=i<=N$ is represented by a set of L_i wrapper configurations, $Z_{i1}, Z_{i2}, \ldots Z_{ij}, \ldots Z_{iLi}$ given as pairs (W_{iZj}, $T(W_{iZj})$). Find a partition of *K* among *B* TAMs, an assignment of cores to TAMs, wrapper design for each core, an assignment of wrapper pins to the pins of SoC and determine the test starting time for all cores such that the overall test time is minimized, and the power budget *Q* is not exceed during the entire testing

8. Experimental Results

All of our tests were conducted on a SUN UltraSPARC 5. We used the ITC'02 SOC benchmarks [13] and we averaged all results over 100 runs. Table 1 shows a comparison of our test-time results with previously published methods. Numbers in bold represent the best results. EA[C] and EA[nC] indicate our results for consecutive and non-consecuitve pins, respectively.

Our results significantly improve upon earlier approaches and are generally better than the results in [14]. It should be noted that [14] used a heuristic from [8] to generate initial solutions for simulated annealing., despite which, CPU time reported in [14] was between a few seconds and two minutes whereas our CPU time was less than 50 seconds for all ITC'02 benchmarks, even though we started with random solutions. In [8] it was indicated that all results were generated in less than one second. For U226 we observed some discrepancy between our results and those of [8] and [14]. We have carefully checked the wrapper design for that SOC and noticed that for core 4 in that benchmark the best possible test time for 10 or more SOC pins is 5333 clock-cycles, therefore it seems impossible to achieve overall testing time of 4080 clock-cycles, unless a different approach was used for wrapper design in [8] and [14].

Table 2 shows results for h953 with power constraints. To compare with [8] we used the same 3 power limits for the SOC power budget, $6•10^9$, $7•10^9$, and $8•10^9$, choosen above the maximum power consumed by one of h953 cores ($5.75•10^9$). These results are given in column (P=const). For all three experiments we generated exactly the same results as in [8], which is probably due to a very restricted power limits. It can be observed that the results do not depend on the number of SoC TAM wires and depend very little on power limits. In the second experiment we assumed that the peak power dissipated during a test is a linear function of the number of TAM wire used. The results of these cases are given in column (P= f (TAM)). We assumed the power numbers given for the benchmark h953 are for a single TAM wire used. So, for example, for 4 TAM wires used, power consumed by the core during testing, given in the benchmark, was multiplied by 4. In Table 3 we compare our multi-TAM results to those from [5]. For most of the cases our results are better, even though those in [5] were obtained by enumeration of all possible cases.

9. Conclusions

We proposed an EA-based approach that uses sequence pair representation to simultaneously perform wrapper design, test scheduling, and assignment of wrapper pins to SOC pins to minimize test application time for a given SOC without exceeding the SOC power budget. We have incorporated a distributed rectangle approach to allow non-consecutive SOC pins to be assigned to a given core for testing. Heuristic modifications to EA operators were developed and shown to generate good results

Power dissipated during the testing was calculated and included into the cost function.. Based on the results for power-constraint test scheduling it seems likely that power limitation might be a stronger limitation than the number of TAM wires. It was also shown that EA is a feasible approach to solving SoC test-scheduling problems with multiple TAMs.

References

1. V. Iyengar, K. Chakrabarty, E. J. Marinissen, "On using Rectangle Packing for SOC Wrapper/TAM Co-Optimization," *Proc. of VTS'02*, pp. 253-258, 2002.
2. K. Chakrabarty,, "Test Scheduling for Core-based Systems Using Mixed-Integer Linear Programming," *IEEE TCAD, pp.1163-1174, 2000.*
3. V. Iyengar, K. Chakrabarty, E. J. Marinissen, "Wrapper/TAM co-optimization, constraint-driven test scheduling, and tester data volume reduction for SOCs," *DAC'02, pp. 685-690, 2002.*
4. Y. Zorian, E.J.Marinissen, S. Dey, "Testing Embedded-Core-Based System Chips", *Proc. ITC'98*, pp. 130, 1998.
5. V. Iyenger, K. Chakrabarty, E. J. Marinissen, "Test wrapper and test access mechanism co-optimization for system on a chip design", *Proc. of ITC'01*, pp. 1023, 2001.
6. S. K. Goel, E. J. Marinissen, "effective and efficient test Architecture Design for SoCs, " *ITC'02, pp.529-538. 2002.*
7. S. Koranne and V. Iyengar, "On the Use of k-tuples for SoC test Schedule Representation," *Proc. ITC'02*, pp.539, 2002.
8. Y. Huang et al., "Optimal Core Wrapper Width Selection and SOC Test Scheduling Based on 3-D Bin Packing Algorithm," *ITC'02*, pp. 74, 2002.
9. K. Chakrabarty, "Design of System-on-a-Chip Test Access Architectures under Place-and-Route and Power Constraints," *DAC'00, pp. 432, 2000.*
10. H. Murata, K. Fujiyoshi, S. Nakatake, and Y. Kajitani, "Rectangle packing based module placement", *Proc. ICCAD*, pp. 472-479, 1995.
11. X. Tang and D. Wong, FAST-SP: "A Fast Algorithm for Block Placement Based on Sequence Pair," *ASP-DAC*, pp. 521-526, 2001.
12. D. Fogel, "Evolutionary Computation: Toward a New Philosophy of Machine Intelligence*", IEEE Press,* 2000.
13. E. J. Marinissen, V. Iyengar, K. Chakrabarty. ITC'02 SoC Test Benchmarks http://www.extra.research.philips.com/itc02socbench/.
14. W. Zou , S. R. Reddy, I. Pomeranz, Y. Huang, "SOC Test Scheduling Using Simulated Annealing", *VTS 2003*.

Table 2. Test-Time under Power Constraint Results for h953

Power Limit	**Number of TAM wires**	
	16-64 (P=const)	**16–64 (P= f (TAM))**
5,753,800,192	122636	489945
$6 * 10^9$	122636	489945
$7 * 10^9$	119357	368191
$9 * 10^9$	119357	257151
$21 * 10^9$		120861
$30 * 10^9$		119357

Table 3. Multi-TAM partitions for D695 benchmark:

	[5]			**EA**			**[5]**			**EA**		
TAMs	**K1+K2**	**T**	**CPU (s)**	**K1+K2**	**T**	**CPU (s)**	**K1+K2+K3**	**T**	**CPU (s)**	**K1+K2+K3**	**T**	**CPU (s)**
16	6+10	43238	1	5+11	**41983**	42	3+5+8	**41949**	4	4+3+9	42424	42
24	6+18	28626	1	9+15	**28235**	25	2+5+17	**28327**	10	1+14+9	29141	25
32	11+21	24030	1	15+17	**21261**	25	4+10+18	**21423**	16	9+6+17	21518	29
40	8+32	20815	2	19+21	**17209**	24	4+17+19	**17210**	24	13+18+9	17565	27
48	16+32	18911	2	19+29	**14418**	27	11+18+19	16403	40	36+2+10	**14456**	32
56	19+37	17671	2	17+39	**12134**	39	4+18+34	13023	53	21+19+16	**12192**	23
64	20+44	17375	3	20+44	**10869**	42	5+20+39	12327	84	4+23+37	**10869**	25

Table 1. Test Scheduling Time for ITC'02 SOC Benchmarks

ITC'02 Bench mark	Solution	**Number of TAM wires**								
		16		**24**	**32**	**40**	**48**	**56**	**64**	
		Test Time (Clock Cycle)	CPU Time [s]	Test Time (Clock Cycle)	Test Time (Clock Cycle)	Test Time (Clock Cycle)	Test Time (Clock Cycle)	Test Time (Clock Cycle)	Test Time (Clock Cycle)	CPU Time [s]
D695	EA [C]	41553	4	**27982**	**21014**	**16908**	14240	**11988**	**10571**	4
10cores	EA [nC]	41809	9	27989	21142	17015	14236	12134	10788	7
	SA [14]	**41604**		28064	21161	16993	**14183**	12085	10723	
	[1]	43723		30317	23021	18459	15698	13415	11604	
	[5]	41949	26	28327	21423	17210	16403	13023	12327	290
	[6]	44307		28576	21518	17617	14608	12462	11033	
	[7]	46152	121	30777	22669	19551	16388	13877	11893	121
	[8]	42716		28639	21389	17366	15142	13208	11279	
P93971	EA [C]	**1754980**	14	1171190	886038	**706820**	600986	**501057**	**445748**	32
32cores	EA [nC]	**1754980**	42	1184630	900388	724758	611029	520868	458389	43
	SA [14]	1757452		**1169945**	**878493**	718005	**594575**	509041	447974	
	[1]	1851135		1248795	975016	794020	627934	568436	511286	
	[5]	1775099	42	1192980	899807	705164	602613	521806	463707	440
	[6]	1791638		1185434	912233	718005	601450	528925	455738	
	[7]	2404321	600	1598829	1179795	1060369	717602	625506	491496	600
	[8]	1791860		1200157	900798	719880	607955	521168	459233	
P34392	EA [C]	**939855**	8	641514	544579	544579	544579	544579	544579	4
21 cores	EA [nC]	**939855**	17	637263	544579	544579	544579	544579	544579	13
	SA [14]	944768		**628602**	544579	544579	544579	544579	544579	
	[1]	1023820		759427	544579	544579	544579	544579	544579	
	[5]	998733	222	720858	591027	544579	544579	544579	549579	1122
	[6]	1010821		680411	551778	544579	544579	544579	544579	
	[7]	1191289	300	838643	568133	544579	544579	544579	544579	300
	[8]	1016640		681745	533713	544579	544579	544579	544579	
P22810	EA [C]	438783	21	292824	226545	**167792**	153260	133094	117638	13
30cores	EA [nC]	**438619**	38	**289237**	228732	183133	153525	130949	116625	39
	SA [14]	**438619**		289287	**218855**	175946	**147944**	**126947**	**109591**	
	[1]	452639		307780	246150	197293	167256	145417	136941	
	[5]	462240	18	361576	312662	278360	268474	266800	260639	120
	[6]	458068		299718	222471	190995	160221	145417	133405	
	[7]	541402	300	356039	294788	220674	203257	175946	157527	300
	[8]	446684		300723	223462	184951	167858	145087	128512	
U226	EA [C]	**13333**	2	10666	8084	8000	7999	5333	5333	2
5cores	EA [nC]	**13333**	5	10666	8084	8000	7999	5333	5333	5
	SA [14]	**13333**		**8084**	**6746**	**5332**	**5332**	**4080**	**4080**	
	[6]	18663		13331	10665	8084	7999	7999	7999	
	[8]	13416		10750	6746	5332	5332	4080	4080	
A586710 5cores	EA [C]	**31877284**	3	**22973206**	**17126866**	**13522326**	**12700205**	**11486599**	**9572166**	3
	EA [nC]	32626780	6	**22973206**	**17126880**	14249800	12811087	11486600	9572170	6
	SA [14]	32626782		23413604	18838663	14260261	12811087	12573448	10659014	
	[6]	41523868		28716501	22475033	19048835	15315476	13401034	12700205	
	[8]	42198943		27785885	21735586	19041307	15071730	14945057	12754584	