INTEGRATION, the VLSI journal 50 (2015) 61–73

Contents lists available at ScienceDirect

INTEGRATION, the VLSI journal

journal homepage: www.elsevier.com/locate/vlsi

Window-based peak power model and Particle Swarm Optimization guided 3-dimensional bin packing for SoC test scheduling ☆

Rajit Karmakar *, Santanu Chattopadhyay

Dept. of Electronics & Electrical Communication Engineering, Indian Institute of Technology Kharagpur, Kharagpur 721302, India

ARTICLE INFO

Article history:
Received 14 March 2014
Received in revised form
23 January 2015
Accepted 24 January 2015
Available online 7 February 2015

Keywords:
System-on-Chip test scheduling
Power-aware testing
Window-based peak power model
Particle swarm optimization
3-D bin packing

ABSTRACT

This paper addresses the issue of power-aware test scheduling of cores in a System-on-Chip (SoC). While the existing approaches either use a fixed power value for the entire test session of a core or cycle-accurate power values, the proposed work divides the power profiles of cores into fixed-sized windows. This approach reduces the number of power values to be handled by the test scheduling algorithms while reducing the amount of pessimistic over-estimations of instantaneous power consumption. As a result, the power model can be integrated with more exhaustive meta-search techniques for generating power constrained test schedules. In this paper, the proposed power model has been integrated with a Particle Swarm Optimization (PSO) based 3-dimensional (3-D) bin packing technique to generate test schedules. Experimental results prove the quality of the approach to be high compared to the existing scheduling techniques.

1. Introduction

SoC paradigm has evolved to address the problem of high design turnaround time of complex VLSI systems. Design time can be reduced by following the core-based system design. Pre-designed logic blocks (processor, memory etc.), commonly known as Intellectual Property (IP) cores, are integrated on the same silicon floor by the system integrator. Compared to a board-based system, in SoC, communication between the system components becomes on-chip. This aids in improving the system performance. However, with rapid growth in VLSI industry, large numbers of IP cores are being integrated onto a single chip. As a result, the amount of test data needed to test a SoC has increased manifold. Automated Test Equipment (ATE), which stores and transports test data to the cores, has to take care of this large volume of test data. Test Access Mechanism (TAM) is a set of wires that makes the physical connection between ATE and cores. TAM is responsible for transporting the test stimuli from the source (i.e. ATE) to the core under test (CUT), and also the responses from the core to the sink (i.e. ATE). Fig. 1 shows a typical example of SoC.

Embedded cores, delivered from different vendor, are commonly accompanied with the associated test vectors to test them. To test a chip, a system integrator applies all the test patterns specified by the vendors of individual cores, increasing Test Application Time (TAT) significantly. Modular testing has been advocated to simplify test access and test application. To facilitate modular test, an embedded core must be isolated from surrounding logic via a test wrapper. A wrapper for a core is a thin shell around the core that acts as a communication bridge between the TAM and the core. It provides width adaptation between core I/O pins and TAM pins.

To keep TAT within a reasonable limit, one solution is to perform parallel testing of several cores. Overlapped testing of cores may again be prohibited by the system power limit. To keep the power budget of a system low, several design techniques, such as, clock-gating [1–5] and power-gating [4,5] are generally adopted. Power is also controlled with the knowledge about the run-time activities of different modules within the system. For power reduction, a designer has to be concerned only with the modules (cores) active simultaneously. However, the test engineer may need to test some modules simultaneously, which are running in an exclusive fashion during normal system operation. Moreover, for a single module, test mode power is much higher than functional mode power. Successive input signals in functional mode are often highly correlated. In test mode, successive test patterns are made as uncorrelated as possible with the expectation that further patterns will excite different regions of the module, possibly identifying newer faults. Thus, in SoC testing, the test engineer is limited by the following.

1. Limited number of channels of ATE to transport test patterns to the chip.

☆This work is partially supported by the research Project no. 9(5)/ 2010-MDD dt. 23/1/2011, sponsored by the DeitY, Govt. of India.

* Corresponding author.
E-mail addresses: rajit@ece.iitkgp.ernet.in (R. Karmakar), santanu@ece.iitkgp.ernet.in (S. Chattopadhyay).

http://dx.doi.org/10.1016/j.vlsi.2015.01.006
0167-9260/

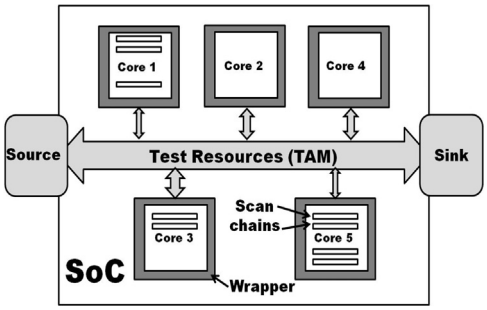

Fig. 1. A typical example of System-on-Chip (SoC).

2. Limited amount of on-chip test resources (TAM wires). It may be a few signal lines running through the chip to carry the test signals.
3. Different number of input–output pins (including scan input/output) for different cores coming from different IP vendors. Scan chain lengths are also varying.
4. Huge amount of test data (pattern and response) to be stored in the ATE and transported to/from the chip.

The problem gets aggravated by the addition of power limitation. A set of cores can be tested in parallel only if it does not violate the system level power limit at every point of the schedule. Determining such a schedule is difficult as it is necessary to ensure power limit validity at every time instant. Conventionally, a global peak power model has been assumed [6–13] for the entire test session of a core. On the other extreme, a cycle-accurate power model has been followed in [14,15]. A global peak power model may generate a too restrictive schedule inhibiting potential parallel testing of some cores, even if the sum of exact instantaneous power values of them is well below the power limit. A cycle-accurate power model does exact power sums; however, the scheduling algorithm needs to handle a large number of power values. It is worth mentioning that excessive power consumption causes overheating, hence may cause permanent damage to the chip. Thus, power validation cannot be ignored. At the same time, using a cycle-accurate power model may become a stumbling block for the scheduling algorithm in search-space exploration.

This paper proposes to use an intermediary approach to handling huge amount of power values for the cores in the scheduling process. For a core and a particular wrapper width, the cycle-accurate power values are obtained. Switching activities in multiple scan chains of cores, caused by the test stimulus and responses during *scan-in*, *launch-and-capture* and *scan-out* operations, have been utilized to obtain the power estimation. However, instead of taking cycle-accurate power, the peak power values over a time-window have been taken to approximate the core power over that interval of test time. The process, though introduces some inaccuracy in the power model, works well for most of the test cases. Moreover, the reduction in the number of power values has also enabled us to design a 3-D bin packing heuristic, guided by the PSO [16] based search procedure, to evolve better test schedules than many of the contemporary SoC testing approaches, while working with ITC'02 benchmarks [17,18]. CPU time needed to generate the test schedule improves by two order of magnitude compared to a similar scheme using the cycle-accurate power model. It may be noted that the approach can also be extended to work in conjunction with variants, such as, multiple voltage islands [19], and voltage and frequency scaling [20–22]. The proposed approach can augment those power models as well, as these techniques assume global peak power values only.

The rest of the paper is organized as follows. Related works have been discussed in Section 2. Section 3 presents the proposal of the window based peak power model. Section 4 illustrates the test scheduling procedure. Experimental results and discussions are noted in Section 5. Section 6 draws the conclusion of the paper and enumerates a few probable extensions.

2. Related work

Power-constrained SoC test scheduling problem is addressed in three steps. In the first step, a wrapper is designed on top of each core to facilitate modular testing and width adaptation between TAM pins and core I/O pins. An efficient wrapper design helps to reduce test time of individual cores. Wrapper design problem was first solved in [23]. The *Design_Wrapper* algorithm has been proposed in [24] to partition the scan chain elements into several wrapper scan chains. Lengths of wrapper chains have been kept as equal as possible to create balanced chains. Authors have also shown that the test application time varies with the TAM width in a "*staircase*" manner. These discrete widths are called *pareto-optimal* points. Test wrapper design for cores in a hierarchical SoC has been presented in [25]. For hierarchical SoC, an embedded core itself can have a core embedded within it. Both the parent and child cores can be tested using different test modes.

After designing optimized wrapper for individual cores to minimize core test times, the major challenge of test scheduling is resource allocation to the cores. Optimized resource allocation and core ordering in schedule play key roles to reduce overall TAT. Resource allocation problem has been approached in two different ways in the literature. In fixed-width TAM architecture [24], total TAM width is partitioned into several test buses of fixed widths. Each core is then assigned to exactly one of the partitioned test buses. The authors in [24] have integrated the wrapper design, TAM partitioning and core scheduling problems into a single one. Fixed width TAM partitioning has been followed by optimal core assignment to individual TAM buses. A similar problem has been solved in [26] using a heuristic method. However, fixed width TAM architecture suffers from the problem of inefficient use of TAM buses, resulting in longer TAT. This shortcoming of resource wastage of fixed width TAM architecture can be overcome using flexible width TAM architecture [27–34], where the total TAM width is not completely partitioned before scheduling. Rather, the resources are allocated to the cores according to the availability and requirement. This flexibility in TAM partitioning explores a huge search space for a reasonably large SoC; hence, there is a chance of getting better schedules, thus, minimizing the TAT. Figs. 2 and 3 show typical examples of two different TAM architectures.

The integrated problem of flexible width TAM partitioning and test scheduling is an NP-hard [28]. It has been solved using different optimization techniques in the literature. An architecture-independent theoretical lower bound on test time has been presented in [35]. The work also presents a heuristic named TR-ARCHITECTURE, suitable for both the fixed and flexible-length scan chains. In [27], the scheduling problem has been formulated as a two-dimensional bin-packing problem. Here, bin height is equal to the total number of available test signal lines. Testing of each core is represented as a

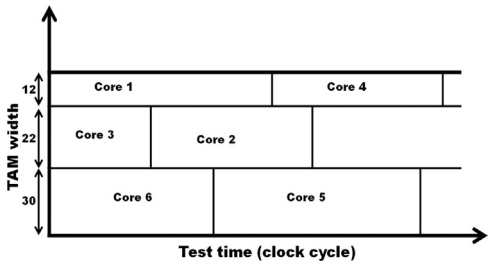

Fig. 2. A typical example of fixed-width TAM architectures.

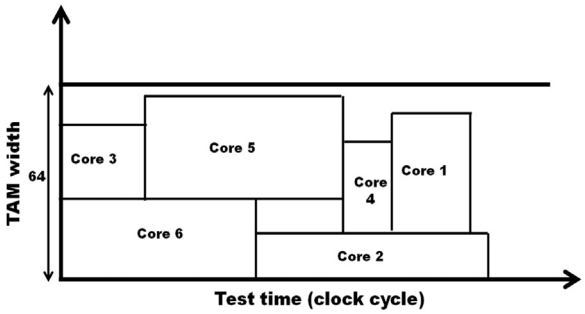

Fig. 3. A typical example of flexible-width TAM architectures.

rectangle having height equal to the number of signal lines allocated and width corresponding to the associated test time. These rectangles are to be packed into the bin so that the width of the bin is minimized. An Evolutionary Algorithm (EA) based approach using sequence pair representation and distributed bin-packing has been presented in [29]. The Simulated Annealing (SA) based two-dimensional bin-packing algorithm to solve SoC test scheduling problem has been proposed in [30]. The work also proposes a wrapper design procedure for cores with no scan cells. The B*-tree based floorplanning technique has been used in [31] transforming the scheduling problem to floorplanning. Height and width of the floorplan represent the TAM width and test time of the core respectively. The idea is to find a fixed height, minimum width floorplan, while there should be no overlap between the tiles in the floorplan. Simulated annealing has been used to solve this problem. The two stage Genetic Algorithm (GA) based approach to solve SoC test scheduling problem has been presented in [32]. Ant Colony Optimization (ACO) has been used in [33] to solve the SoC scheduling problem. The work reported in [34] uses another Genetic Algorithm (GA) based approach to select the rectangles to pack in a bin for the rectangular 2D- bin packing.

It may be noted that, all the above-mentioned approaches assume static TAM assignments, i.e., TAM width is fixed during the test. A reconfigurable core wrapper approach with the flexibility of dynamic change in TAM width during the execution of core test to improve test schedule has been presented in [36]. In [37] reconfigurable core wrappers have been used in SoC test scheduling. Although reconfigurable wrappers lead to efficient test schedules, more gate and routing overheads are imposed. It also increases the control complexity of the wrapper.

Test parallelism also increases power consumption and must be restricted by system level power limit for the safety of the chip. This extra power constraint adds the final dimension to the power-aware test scheduling. Several works in the literature consider power restricted test schedule generation. Different power models of the core have also been proposed. The concept of considering maximum power consumption value of a core as the fixed power value throughout its testing, has been presented in [6–11,13]. This global peak power model, although simple, is quite pessimistic, as power consumption is never same throughout the duration of testing of the core. It may reach its peak at certain time instants only. Also, all the cores being tested in parallel, may not reach their peak power values at the same time instant. The resulting conservative schedule increases the TAT. In [7], the rectangular 3-D bin packing approach has been adopted to solve SoC test scheduling problem under power constraint. Power has been considered as the third dimension with test resource and test time as other two dimensions of the bin. The authors, in [8], have considered multiple test sets for testing of the core. An Integer Linear Programming (ILP) based solution has been presented in [9]. Here the authors have presented a reward model that allows the system integrator to incorporate preferences arising from place-and-route constraints in allocating cores to test buses. A greedy algorithm based technique has been proposed in [10]. A Genetic Algorithm based approach, to minimize TAT has been reported in [11]. Here the authors have used chromosome structure, mutation and crossover operators to select representative rectangles for the rectangular 3-D bin packing approach to solving the scheduling problem optimally. A shuffle frog-leaping algorithm (SFLA) that follows evolution of frog population has been used in [12] to solve the scheduling problem under power constraint. All these power-aware test scheduling techniques have used the global peak power model.

The exact power model of each core has been presented in [14]. The work approximates the power values for each cycle during testing of the core, to be equal to the total transition in the scan cells, during test pattern shift and capture in that particular clock cycle. The scheduling procedure has been implemented using Best-Fit heuristics with fixed width TAM architecture [14,15]. Handling a huge number of power values and keeping track of the power profile for each clock cycle during scheduling makes this approach extremely time-consuming. The authors in [38,39] have tried to reduce power overestimation by a less complex method using test vector reordering, test sequence expansion and rotation. The manipulated power profile obtained using these methods has an initial long low power part followed by a short high power part. However, the use of this two local peak power approximation model (2LP-PAM) is limited to the cases when testing is performed exclusively using ATPG-generated test vectors and where the order of the test vectors can be changed. Unfortunately, in most of the cases this pattern reordering may not be feasible as the ATE is preloaded with test patterns.

To overcome the scheduling complexity of cycle-accurate power model and to reduce the power overestimation of global peak power model, our work considers the window-based power model, which needs reasonably less amount of power values for a core, hence able to generate test schedule much faster than with the cycle-accurate power model. Unlike [14,15] our approach does not partition the total available TAM width into fixed sized buses. It rather follows the bin-packing approach [28] to introduce flexibility in TAM width allocation to cores. The 2-D bin packing strategy of [28] has been extended to 3-D bin packing concept with wrapper dependent window-based peak power as the third dimension. In recent times Particle Swarm Optimization [16] has evolved as a new evolutionary search strategy that performs better than other evolutionary approaches because of its faster convergence towards the optimal solution. We have used the PSO based evolutionary technique to guide the bin-packing algorithm. The existing approaches either use some deterministic heuristic [14,15] with some elaborate power model, or a rigorous optimization strategy (such as ILP [9], ACO [33], and GA [11]) with the fixed power model. As discussed in this section, both have shortcomings. Our approach attempts to bridge the gap by showing the usage of meta-search technique like PSO along with a power model of sufficient detail within a feasible computational overhead.

Several recent works have attempted to solve the test scheduling problem by using different voltage levels and also by varying the operating clock frequency. Multiple voltage testing of cores, where cores are placed in multiple voltage islands, has been presented in [19]. The formulated test scheduling problem has been solved using ILP and also via a greedy approach. Another ILP based method, considering different operating frequencies for different test sessions to minimize test time has been presented in [20]. In [21] both supply voltage as well as, operating frequency have been varied to get better schedule. This dynamic voltage frequency scaling (DVFS) method can be used properly to slow down or speed up individual tests to get better test compatibility under power constraint, during scheduling. A pre-emptive and a non-pre-emptive heuristic approach using the DVFS method for session-less test scheduling has been presented in [22]. It may be noted that the window-based peak power model proposed in our

work can go hand-in-hand with these approaches to exploiting the architectural advances like voltage islands and frequency scaling.

3. Window-based peak power model

During testing of a core, test stimuli shift into the wrapper scan-in chains, launch the test (single capture cycle) and corresponding test responses shift out through the wrapper scan-out chains. Transitions occur in the scan cells at the time of *shift-in*, *launch-and-capture* and *shift-out* operations. While a test response shifts out through the scan-out chains, the next test stimulus shifts into the scan-in chains. Suppose, a certain wrapper configuration of a core C_i has l wrapper chains with wrapper scan-in lengths SI_1, SI_2, $SI_3 \ldots SI_l$ and wrapper scan-out lengths SO_1, SO_2, $SO_3 \ldots SO_l$. The core is tested with p test patterns. Total test time (T) is calculated as [23]

$$T = (1 + \max(SI_i, SO_i)) \times p + \min(SI_i, SO_i) \quad (1 \le i \le l) \tag{1}$$

The length of test stimuli is equal to the sum of primary inputs (*PI*), Bidirectional lines (*Bidir*) and number of scan cells (*SC*), while test responses are the sum of primary outputs (*PO*), *Bidir* and *SC*. The test stimuli and test responses split themselves according to the length of the wrapper-scan-in chains and wrapper-scan-out chains respectively in the case of multiple wrapper-chain configurations. Sometimes we need to pad some extra bits to the test patterns to match with the length of $\max(SI_i, SO_i)$. Power consumption of a core can be described by the following equation:

$$P = 1/2CV_{DD}^2 \alpha f \tag{2}$$

where C is the output capacitance, V_{DD} is supply voltage, α is the number of switching activities in scan chains during test pattern shift and f is the operating frequency. As output capacitance and supply voltage do not change with time and operating frequency also remains unchanged for single frequency operation, the only varying factor is α, which changes with time. This switching activity is the main contributor of power during testing. We consider the total switching activities in wrapper chains in a particular clock cycle as the power consumption of that cycle. However, switching activity depends on the length of the wrapper chains. Different wrapper chain configurations have different switching activities, hence different power profiles. For multiple scan chains, multiple test data are shifted to the scan chains at each clock cycle while only single bit test data is shifted at each cycle for a single scan chain. Obviously, total transition counts in the wrapper chains vary with the number of wrapper chains. Hence, a core has several power profiles corresponding to each wrapper configuration of it. Fig. 4 shows the variation in the power profile with variation in the number of wrapper chains of module number 1 of ITC'02 benchmark circuit, *d*695, when the module is tested with a single test pattern. Transition count increases gradually at the time of *shifting-in* the test stimulus. Abrupt change in the power value indicates the *launch-and-capture* operation. This happens since at this time, some scan flip–flops change their content, getting values from the circuit under test. *Shift-out* operation is clearly indicated by the gradual decrease in transition count.

Cycle-accurate power values can be computed by calculating total transition count in scan chains at each cycle for all sets of wrapper configurations of each core. However, for bigger circuits with longer test time values, it is very difficult to handle a large number of associated power values in the test schedule generation process. From the cycle accurate power profiles, it can also be observed that for most of the time instants, there is no significant variation in the power profile in the neighborhood of it. This has motivated us to make the power-model coarse-grained via windowing.

In the window-based peak power model, we partition the total test time of a core into some smaller sized time windows and consider a single peak power in that interval to represent the power value for that interval. The window-based power model gives us a faster test scheduling strategy than the cycle-accurate power model since now we need to work with less number of power values. A single global peak-power model introduces high amount of overestimation compared to the actual instantaneous power values. The window-based power model has the capability to reduce the amount of this overestimation. Very small window interval leads towards the cycle-accurate power model while large window-size incorporates more false power values in the schedule. Fig. 5 illustrates the concept of window-based peak power model. It shows the cycle-accurate power values, global peak power of 15 units and the window-based model with window-size 10. For example, for the time interval 0 to 10, the peak value is 8 units and thus, it has been used as the power value for the entire window. From the figure, it can be observed that the amount of over-estimation of power is much less in the window-based model compared to the global peak power model. For example in Fig. 5, the number of power values to be remembered is only 5 in the window-based model compared to 50 for the cycle-accurate case. For the full 50 cycle operation, overestimation in the global peak power model is 353 units (computed by summing up the differences between the global peak of 15 units and the instantaneous power values), while in the window-based model, it reduces to 143 units.

The accuracy of window-based peak power model depends on window-size to a large extent. In our formulation, the length of

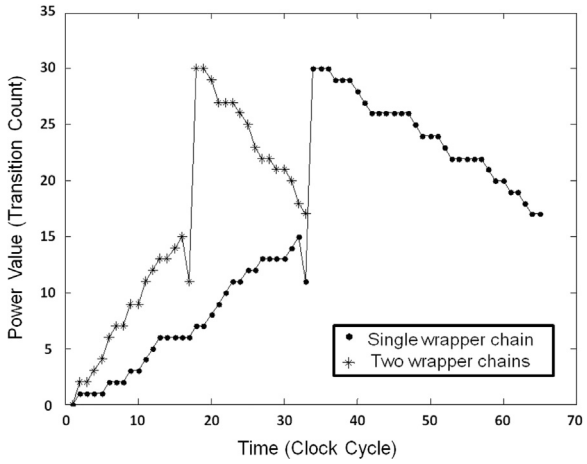

Fig. 4. Power profiles of module no 1 of *d*695 for different wrapper configurations.

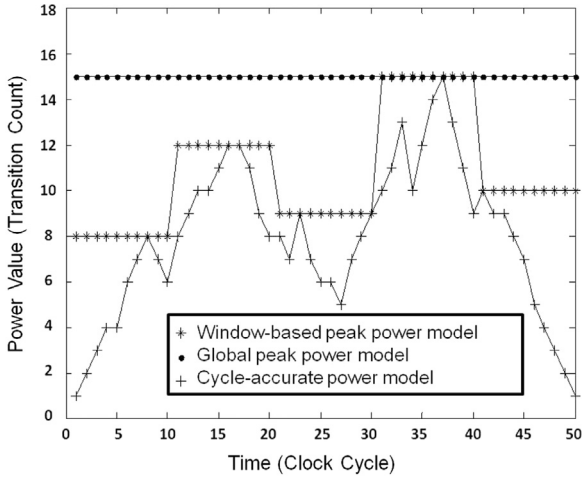

Fig. 5. Comparison of window-based peak power model with other power models.

window interval has been kept flexible and can be tuned according to the requirements and computational resources available. Obviously schedule generation time depends on the length of the window interval. Fig. 6 shows the variation of CPU time required to generate a valid schedule for different window-sizes for ITC'02 benchmark *p*93791 using the test scheduling algorithm Schedule_Rectangle mentioned in Section 4.2.2. It is clear from the figure that CPU time increases exponentially with a reduction of window-size from global peak (GP) to cycle-accurate (CA). However, it is worth mentioning that CPU times reported in Fig. 6 for different window-sizes are the times required only to generate a single valid schedule, which do not ensure near optimal TAT. Generation of near-optimal solution for NP-Hard problems like test scheduling requires some evolutionary technique, where schedule generation time depends not only on scheduling algorithm, but also on the total number of iterations required to evolve a solution. So, final schedule generation time is much higher than the time required for a single solution and may be too costly for the cycle-accurate power model.

The benefit of using the window-based peak power model over the global peak power model can be explained using Figs. 2 and 3. Let us assume that we have three cores core 1, core 2 and core 3 with test times 4700, 1550 and 2680 clock cycles respectively. Global peak power values of the cores are assumed to be 13, 8 and 11 units respectively and the maximum system-level power limit is 22 units. We tried to schedule the cores using both global peak power model and window-based peak power model. Fig. 7 shows the scheduling of the cores using the global peak power model while Fig. 8 depicts the test schedule using the window-based peak power model. We have considered window-size of 1000 clock cycles for the window-based peak power model. Window-based peak power value for core 1 in the window interval 1 to 1000 is assumed to be 10 units while these values are 7, 13, 10 and 4 units for successive window-intervals. It may be noted that core 1 attains its peak power in the time interval 2000 to 3000 clock cycles while its power values are less for other window-intervals. Similarly, the window-based peak power values for core 2 and 3 are pictorially described in Fig. 8. Now, core 1 and core 3 cannot be scheduled in parallel using the global peak power model (Fig. 7), as the sum of their peak power values (i.e. 13+11=24) violates the system level power limit value of 22 units. To satisfy the power constraint, we have to serialize the test schedule by scheduling core 3 only after core 1 finishes, making the TAT unnecessarily longer. However, it may be noted from Fig. 8 that core 1 and core 3 do not attain their respective peak power values at the same time. The window-based peak power values of core 3 in the interval 2000 to 3000 clock cycles (where core 1 attains its peak power value of 13 units) are 6 and 5 units. Similarly, core 3 attains its peak power value of 11 units in the time interval 3550 to 4230 clock cycles, where the window-based peak power values of core 1 are 10 and 4 units. It is clear from Fig. 8 that, there is no power limit violation even if we test cores 1 and 3 in parallel using the window-based peak power model. This reduces the TAT by reducing unnecessary power over-estimations and increasing test concurrency over the global peak power model.

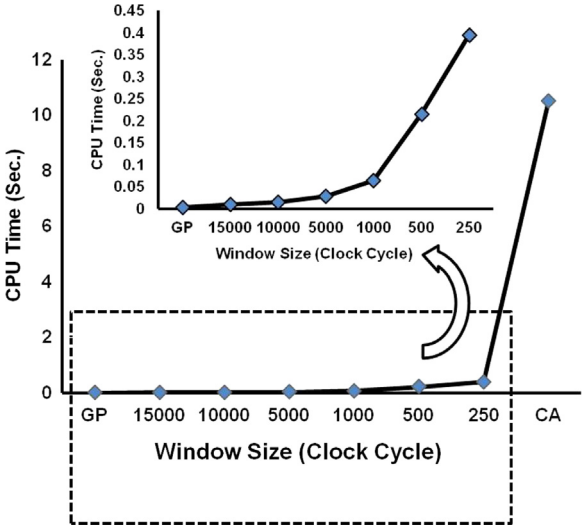

Fig. 6. Variation of schedule generation time with window-size for *p*93791 for a single iteration.

4. Test scheduling

4.1. Problem formulation

Suppose a SoC with N cores C_1, $C_2 \ldots C_N$ is to be tested with a maximum of W_{max} TAM resources and a maximum power limit P_{max}. The test scheduling problem is to allocate TAM resources and test times to the cores so that, the total test application time is

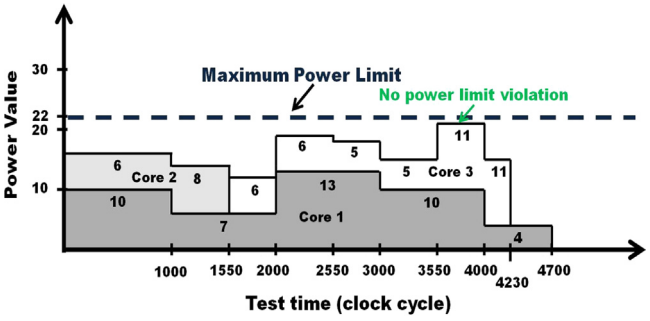

Fig. 8. Test schedule using window-based peak power model.

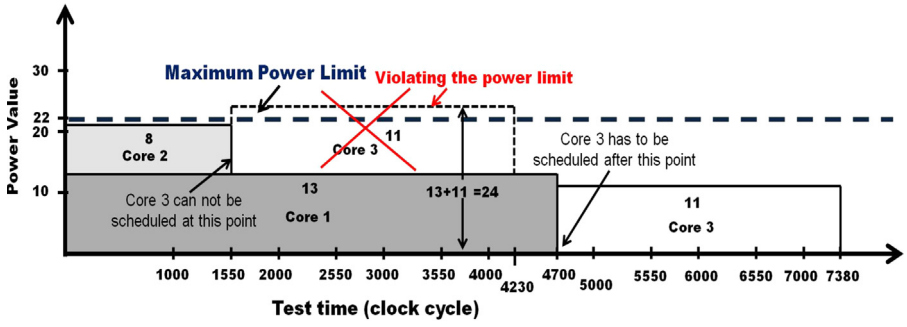

Fig. 7. Test schedule using global peak power model.

minimized and the power consumed by the SoC at each instant of time during test is less than P_{max}.

Due to its flexibility of TAM width attachment to individual cores, rectangular 2-D bin packing has evolved to be a popular method to solve the test scheduling problem for embedded cores [28]. Each core C_i $(1 \le i \le N)$ is represented by a set of wrapper configurations R_i. The test resource requirement of core C_i with jth wrapper configuration can be represented by a rectangle whose height and width represent allocated TAM width (w_{ij}) and the corresponding test time $(T(w_{ij}))$ respectively. To get a schedule for the full SoC, the rectangles are to be packed into a bin of fixed height (W_{max}) so that TAT (width of the bin) is minimized. Power constrained scheduling takes this problem to a 3-D bin packing problem, where power represents the third dimension. We have used the window-based peak power profile for each wrapper configuration of each core. Bin packing is NP-Hard [28]. In the following, we present a PSO based approach to solve the scheduling problem.

4.2. Test schedule generation

The process consists of the following components.

- Generation of test rectangles for individual cores.
- Selecting one test rectangle for each core.
- Scheduling the selected test rectangles.

The first step is a stand-alone one, while the next two stages need to be solved in an integrated fashion. We have used the *Design_Wrapper* algorithm [24] to generate different wrapper configurations for each core. For a core, the wrapper configurations corresponding to only the pareto-optimal TAM widths [24] are noted. The corresponding window-based power profiles are determined. Selection of one test rectangle per core has been performed using PSO. Each particle gives a set of rectangles, one for each core. Fitness of the particle has been evaluated by performing a scheduling of these rectangles.

4.2.1. Particle swarm optimization formulation

PSO is a population-based evolutionary technique designed by Eberhart and Kennedy [16]. It starts with an initial population of particles. Each particle corresponds to a solution to the optimization problem being solved. It has its fitness value. Particles evolve over generations guided by self- and group-intelligence, and also via their inertia. Any PSO formulation involves choosing a proper representation of the particles, their fitness calculation, and defining an evolution policy. Next, we elaborate each of these in the context of power-aware test scheduling of cores in a SoC.

Particle structure: Each core has a set of test rectangles. It may be noted that considering the power values; we can say that each core has a set of test cuboids associated with it – the dimensions being wrapper width, test time, and power. However, since we are not considering pattern reordering, the two cuboids of a core cannot differ only in their power values – we will conveniently call them rectangles only. Let the number of cores in the SoC be N, and the maximum number of rectangles for any core be M. Let $B = \lceil \log_2^M \rceil$. A particle consists of $N \times B$ number of bits. First B bits identify the test rectangle selected for the first core, second B bits for the second core, and so on. Fig. 9 shows a sample particle with $N=4$ and $B=4$. In this case, test rectangles 9, 2, 8 and 13 are selected for cores 1, 2, 3 and 4 respectively. For the initial generation, particles are generated randomly; however care has been taken to ensure that the indices generated for a core are always within the total number of rectangles of it. Fitness of a particle is equal to the total test time of the SoC after scheduling the test rectangles using the procedure noted in Section 4.2.2.

Evolution of a particle: In a PSO formulation, evolution of a particle is guided by three factors – its own intelligence, global (swarm) intelligence, and the inertia factor. A particle always remembers its history about its best structure over generations. This is called the local best (*pbest*) of the particle. In a particular generation, the particle with the best fitness value is the global best (*gbest*) of the generation. For the initial generation, *pbest* of each particle is initialized to itself while the *gbest* of the generation is the best one of the first generation. In the successive generations, new particles are created using the *replace* operator noted next.

The *replace* operator attempts to align a particle with its *pbest* and the *gbest* particles, with some probability. For the sake of this alignment, the *replace* operator is applied at each bit position of a particle. For the bit position i of a particle, the bit is replaced by the corresponding bit of *pbest* particle with probability α. After the operator has been applied for *pbest*, the same is done with reference to *gbest* with probability of replacement, β. After both the replacement operators have been applied to all bit positions for a core, a consistency check is performed. If the new rectangle number for the core becomes larger than the total number of rectangles available for the core, the rectangle number is reverted back to its value in the original particle. In our experimentation, we have kept both α and β values at 0.1. Fig. 10 shows an example alignment of a particle towards its local best.

4.2.2. Scheduling of test rectangles

The algorithm takes as input the rectangle set corresponding to a particle, the maximum TAM width W_{max} and the maximum power limit P_{max}. It performs scheduling of the rectangles, honoring the constraints that at no instant of time, the total TAM width requirement exceeds W_{max}, and the instantaneous power value does not exceed P_{max}. The resulting total test time is the fitness of the particle. At any point of time, the algorithm maintains the

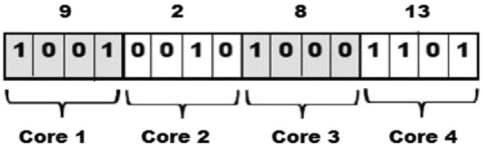

Fig. 9. Sample particle structure of 4 cores with $W_{max}=16$ $(B=4)$.

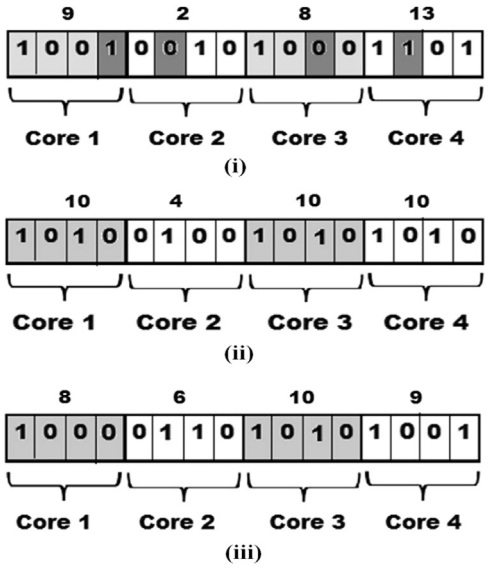

Fig. 10. (i) current particle, (ii) local best and (iii) evolved particle.

following data structures to arrive at a decision about scheduling the next core.

Break_Point_List (*BP*): A set of time instants at which the power requirement of the schedule has changed from its value in the previous instant. The next core can be scheduled at any of the breakpoints, $bp_k \in BP$.

Available_TAM_Width_Info (*ATW*): A set with cardinality same as *BP*. The value atw_k is equal to the total free TAM width available at break point instant bp_k.

Power_Tracker (*PT*): This is also a set with cardinality same as *BP* and holds the total power consumed by the already scheduled cores at the corresponding breakpoint instant.

As the till unscheduled cores get scheduled, the list *BP*, *ATW* and *PT* also get updated. The bin packing procedure also needs to prioritize the next unscheduled rectangle to be selected for packing (scheduling). For this purpose, the rectangles are sorted on their area values (TAM width (w) $\times$ test time (T)) in a descending order. The break-point list *BP* is scanned from the minimum to the maximum value. To make the schedule compact, we try to utilize any available TAM resource and power budget at every break-point. Hence, for the break-point bp_k, the algorithm scans the unscheduled rectangle list to check for the largest rectangle that can be scheduled at bp_k. If none are feasible, the algorithm advances to the next break-point. Power requirements are also checked to ensure satisfaction of power limit at every break-point till the end of the schedule for the current core. When rectangles corresponding to all cores have been scheduled, the maximum end time of testing of all cores gives the total test application time for the SoC. The algorithm to produce the schedule is presented next.

Algorithm 1. *Schedule_Rectangles*.

Input: List of rectangles to be scheduled; W_{max}, the maximum TAM width; P_{max}, the power limit
Var: *BP*: A list of breakpoints; *ATW*: List of available TAM widths at each break point $bp \in BP$; *PT*: List of total power values at each break point $bp \in BP$;

```
1  begin
2    BP ⟵ 0;
3    ATW ⟵ W_max;
4    PT ⟵ 0;
5    Sort list of rectangles on decreasing area;
6    Mark all rectangles as unscheduled;
7    while there exists unscheduled rectangles do
8      while all entries of BP not checked do
9        Let bp_k be the next entry of BP;
10       atw_k ⟵ ATW[bp_k];
11       pt_k ⟵ PT[bp_k];
12       Check if any unscheduled rectangle picked up in sorted order can be scheduled at bp_k with available TAM resource and power budget;
13       if yes then
14         Update BP, ATW, PT and Rectangle List;
15         Mark corresponding rectangle scheduled;
16       end
17       else
18         Continue with next bp_k ∈ BP;
19       end
20     end
21   end
22   Return the maximum test end time among all rectangles.;
23 end
```

5. Experimental results

In this section we present results of experimentation with ITC'02 [17,18] benchmark SoCs. As noted in Section 2, most of the power-constrained SoC test scheduling approaches consider the global peak power model, while few others consider the cycle-accurate power model. However, in our formulation, our main focus is to divide the total test time into some window-intervals and consider peak power values for all those intervals (i.e. window-based peak power model). As our approach does not consider the same power profile suggested in the literature, it is not justifiable to have a direct comparison of our results with those presented in the literature. To have a better comparative analysis, we have divided our results into two subsections. The first subsection considers the same power model (i.e. global peak power model) suggested in the literature and compares the quality of our PSO guided test scheduling algorithm with other solution strategies reported in the literature. The second subsection compares the quality of our proposed window-based peak power model with global peak as well as TAM dependent peak power models. It also performs comparison with cycle-accurate power model results, in terms of solution quality and CPU time requirements.

5.1. Test scheduling with global peak power model

Since many of the power-aware SoC test scheduling approaches consider a single power value for the entire duration of testing a core, first we show results of our approach using this global peak power model. It may be noted that, only SoC *h*953 has

been provided with a list of global peak power values of the cores in ITC'02 benchmark suite. No information regarding power values of the rest of the SoCs are available in the benchmark suite. However, global peak power values of the cores of SoC *d*695, *p*22810 and *p*93791 have been reported in [8]. We have considered these values for the sake of experimentation. We could not carry on the comparison on other benchmarks as no reported data regarding test scheduling under global peak power is available in the literature. Tables 1, 2, 3 and 6 note the power-aware test scheduling results for the benchmarks *h*953, *d*695, *p*22810 and *p*93791 respectively, corresponding to different power limits. While the works reported in [7,8,11,13,12] consider only this global peak power values, [14,15] have reported scheduling results for both global and cycle-accurate power models. Our results noted under the column marked "PSO GP" are better than other techniques including the cycle-accurate model of [14,15], for most of the cases. It may be noted that, benchmark *h*953 does not show any improvement in result with the increase in the available test

Table 1
Comparisons of PSO with other scheduling procedure for *h*953 with different P_{max} and W_{max} (considering global peak power model).

*h*953	**EA** [29]	**3D-Bin** [7]	**GA** [11]	**PSO GP**
W_{max}	$P_{max}=6\times10^9$			
16	122,636	122,636	**119,357**	**119,357**
24	122,636	122,636	**119,357**	**119,357**
32	122,636	122,636	**119,357**	**119,357**
40	122,636	122,636	**119,357**	**119,357**
48	122,636	122,636	**119,357**	**119,357**
56	122,636	122,636	**119,357**	**119,357**
64	122,636	122,636	**119,357**	**119,357**
	$P_{max}=7\times10^9$			
16	**119,357**	**119,357**	**119,357**	**119,357**
24	**119,357**	**119,357**	**119,357**	**119,357**
32	**119,357**	**119,357**	**119,357**	**119,357**
40	**119,357**	**119,357**	**119,357**	**119,357**
48	**119,357**	**119,357**	**119,357**	**119,357**
56	**119,357**	**119,357**	**119,357**	**119,357**
64	**119,357**	**119,357**	**119,357**	**119,357**
	$P_{max}=8\times10^9$			
16	**119,357**	**119,357**	**119,357**	**119,357**
24	**119,357**	**119,357**	**119,357**	**119,357**
32	**119,357**	**119,357**	**119,357**	**119,357**
40	**119,357**	**119,357**	**119,357**	**119,357**
48	**119,357**	**119,357**	**119,357**	**119,357**
56	**119,357**	**119,357**	**119,357**	**119,357**
64	**119,357**	**119,357**	**119,357**	**119,357**

Table 2
Comparisons of PSO with other scheduling procedure for *d*695 with different P_{max} and W_{max} (considering global peak power model).

***d*695**	**MC** [8]	**3D-Bin** [7]	**Cycle accurate** [14,15]		**GA** [11]	**ACO** [13]	**SFLA** [12]	**PSO**
			GP	**CA**				**GP**
W_{max}	$P_{max}=$ **1500**							
16	43,541	45,560	47,009	44,936	**42,189**	42,658	41,226	42,268
24	32,663	31,028	31,458	30,663	30,054	**29,401**	30,418	29,571
32	26,973	27,573	27,544	23,169	23,297	23,267	23,076	**23,059**
40	24,369	20,914	23,937	19,200	**18,883**	–	–	19,277
48	23,425	20,914	20,842	**17,013**	17,083	18,597	18,348	18,604
56	19,402	16,841	18,909	**15,230**	15,670	15,856	16,727	16,191
64	19,402	16,841	16,875	**12,941**	17,695	–	–	16,191
	Pmax = **2000**							
16	42,450	43,221	44,870	44,502	41,804	41,977	**41,295**	41,771
24	29,106	29,419	30,926	30,506	**27,999**	28,389	28,951	28,211
32	21,942	24,171	25,048	22,544	21,592	21,524	23,076	**21,375**
40	18,691	19,206	20,925	18,799	**17,067**	–	–	17,139
48	17,467	17,825	18,553	16,506	15,652	**15,625**	17,186	15,872
56	14,563	14,128	17,013	13,185	**12,987**	**12,987**	14,945	13,267
64	14,469	14,128	14,397	**11,526**	14,434	–	–	13,778
	Pmax = **2500**							
16	41,847	43,221	44,502	44,502	41,718	41,872	**41,295**	41,740
24	29,106	29,023	30,926	30,336	**27,999**	28,289	41,295	**27,999**
32	21,931	23,721	23,525	22,544	21,236	21,484	23,076	**21,042**
40	18,691	19,206	18,988	18,799	**16,965**	–	–	17,071
48	17,257	15,847	16,506	16,506	**14,434**	14,974	17,211	14,896
56	13,963	14,128	14,834	13,185	12,862	12,877	**12,327**	12,826
64	13,394	12,993	13,098	**11,526**	11,646	–	–	11,985

Table 3
Comparisons of PSO with other scheduling procedure for *p*22810 with different P_{max} and W_{max} (considering global peak power model).

*p*22810	MC [8]	Cycle accurate [14,15]		GA [11]	ACO [13]	SFLA [12]	PSO
		GP	CA				GP
W_{max}	$P_{max} = 3000$						
16	482,963	664,511	536,978	431,475	443,285	**394,466**	427,890
24	392,525	607,451	395,389	290,113	295,752	291,545	**288,228**
32	309,255	543,358	349,530	222,095	231,924	246,524	**221,770**
40	356,215	427,876	314,067	181,489	–	–	**180,979**
48	311,632	363,299	287,642	156,041	160,936	161,449	**155,450**
56	293,528	362,487	280,548	138,346	148,143	149,060	**134,486**
64	293,021	350,162	280,548	120,414	–	–	**118,626**
	Pmax = **5000**						
16	472,026	504,509	458,812	428,852	436,930	433,920	**425,113**
24	382,507	365,562	331,169	286,352	296,466	287,967	**285,086**
32	321,930	320,386	275,106	217,083	225,300	234,565	**215,940**
40	264,038	289,649	240,829	**175,946**	–	–	**175,946**
48	266,166	229,998	225,179	**147,898**	152,414	165,436	148,268
56	257,600	219,244	215,183	**127,382**	133,101	147,483	127,591
64	246,110	219,244	197,593	116,625	–	–	**115,628**
	Pmax = **10, 000**						
16	473,418	450,546	450,051	–	436,730	433,920	**425,113**
24	352,834	329,419	308,374	–	293,783	287,967	**284,701**
32	236,186	260,711	241,841	–	225,300	220,594	**214,934**
40	195,733	241,182	199,748	–	–	–	**175,946**
48	159,994	216,632	179,482	–	151,256	160,605	**147,171**
56	138,542	207,606	166,585	–	131,821	147,483	**126,570**
64	128,332	185,309	160,485	–	–	–	**116,625**

resources, although it is expected to have better TAT for higher values of W_{max}, which encourages more test concurrency. This is because all cores of *h*953 have *pareto-optimal* points with lesser TAM values. Individual test times of the cores do not improve with higher assigned TAM. However, other three benchmarks show a significant improvement in TAT for higher resource values. The fact that the test time results in "PSO GP" are often better than cycle-accurate power model based results of [14,15] indicates the quality of the formulated PSO. It also encourages us to see the effect of using PSO under more elaborate power models, such as, window-based one, suggested in this paper.

5.2. Test scheduling with window-based peak power model

Working with the window-based power model requires detailed knowledge about the test patterns and the corresponding core power profiles. In the absence of this information for the benchmarks, we have randomly generated the test pattern sets for cores, guided by the number of inputs, bidirectional lines, outputs, scan cells. However, ITC'02 benchmark *d*695 is formed using some of the ISCAS85 and ISCAS89 circuits as cores. Test patterns of these cores can be generated using any test pattern generator tool. So, instead of generating test patterns randomly, we have generated the test patterns of the cores of *d*695 using the Synopsys TetraMax ATPG tool [40]. We have noted that the number of test patterns generated for each core using this tool is different from the number of test patterns mentioned in the benchmark information. This is because different test pattern generator tools generate different number of test patterns for a particular circuit. Table 4 reports the core details of SoC *d*695 and the number of test patterns for each core, generated using TetraMax. For each core, appropriate numbers of scan chains have been generated and transitions in them for the test sets have been calculated for each cycle. The total number of scan transitions in the cycle has been taken as a measure of power consumption by the core at that cycle. Since power profiles are also dependent on the wrapper configurations, we have also computed the global peak values for individual wrapper configurations of cores. This we call TAM width dependent, global peak power (marked as TP in Tables 7 and 8). Window-based peak power is marked as WP in the tables. For different power limits, test time results have been noted in Tables 7 and 8, for different SoC benchmarks. Here also, in most of the cases, results corresponding to the window-based peak power

Table 4
Core information of *d*695 used in window-based peak power model.

Core no.	Core details	No of test patterns
1	*c*6288	30
2	*c*7552	219
3	*s*838	152
4	*s*9234	146
5	*s*38584	145
6	*s*13207	273
7	*s*15850	130
8	*s*5378	123
9	*s*35932	21
10	*s*38417	99

Table 5
Comparison of CPU time for different power models for all ITC'02 benchmarks (single particle, single generation).

Circuit name	CPU Time (s)			Window size
	GP	WP	CA	(Clock cycle)
*d*695	0.002	0.006	0.279	1000
*p*22810	0.002	0.031	12.279	1000
*p*93791	0.002	0.028	17.386	5000
*p*34392	0.002	0.027	10.343	1000
*g*1023	0.002	0.005	0.421	1000
*t*512505	0.004	0.035	13.724	5000
*q*12710	0.003	0.021	1.24	1000
*f*2126	0.002	0.012	0.341	1000
*d*281	0.002	0.009	0.327	1000
*u*226	0.001	0.005	0.179	1000
*h*953	0.002	0.018	7.42	1000
*a*586710	0.003	0.032	9.21	5000

Table 6
Comparisons of PSO with other scheduling procedure for *p*93791 with different P_{max} and W_{max} (considering global peak power model).

*p*93791	**MC** [8]	**Cycle accurate** [14,15]		**GA** [11]	**ACO** [13]	**SFLA** [12]	**PSO**
		GP	**CA**				**GP**
W_{max}	$P_{max} = 10,000$						
16	1,827,819	–	–	1,770,954	1,815,761	1,803,224	**1,744,001**
24	1,220,469	–	–	1,192,015	1,227,691	1,210,668	**1,175,050**
32	1,117,385	–	–	1,033,988	1,031,103	1,222,874	**980,153**
40	1,091,210	–	–	**841,594**	–	–	871,920
48	691,866	–	–	609,751	639,255	626,725	**606,544**
56	629,051	–	–	573,720	573,720	573,720	**556,735**
64	568,734	–	–	552,410	–	–	**441,808**
	$P_{max} = 20,000$						
16	1,827,819	1,835,416	1,829,232	1,759,656	1,787,856	**1,660,342**	1,730,385
24	1,220,469	1,233,680	1,233,716	1,174,517	187,161	1,203,156	**1,159,733**
32	957,921	932,323	934,069	886,869	912,503	993,822	**871,780**
40	821,575	766,353	769,378	712,053	–	–	**702,252**
48	658,132	640,602	640,615	600,632	623,013	611,527	**583,762**
56	549,669	550,636	539,815	508,947	573,720	598,228	**507,406**
64	493,599	485,297	492,463	450,977	–	–	**444,511**
	$P_{max} = 25,000$						
16	1,827,819	1,829,176	1,829,232	1,756,326	–	–	**1,730,030**
24	1,220,469	1,233,680	1,233,716	1,173,939	–	–	**1,155,331**
32	965,383	929,974	934,069	883,885	–	–	**868,726**
40	821,475	748,140	748,154	708,150	–	–	**697,041**
48	639,217	612,046	625,476	599,339	–	–	**580,743**
56	549,669	545,322	539,815	508,437	–	–	**500,678**
64	493,599	485,297	481,893	449,657	–	–	**438,386**

Table 7
Comparisons of GP and TP with WP using PSO for different ITC'02 SoCs for different P_{max} and W_{max}.

W_{max}	**GP**	**TP**	**WP**	**GP**	**TP**	**WP**	**GP**	**TP**	**WP**	
*d*695	$P_{max} = 1800$			$P_{max} = 2000$			$P_{max} = 2500$			Pmax = ∞
16	58,141	57,821	**57,321**	57,886	57,645	**57,309**	57,367	57,456	**57,194**	**57,087**
24	39,627	39,170	**38,951**	38,968	38,825	**38,584**	38,799	38,588	**38,569**	**38,469**
32	31,277	30,946	**30,209**	29,331	29,438	**29,331**	29,095	29,095	**28,963**	**28,963**
40	24,145	24,145	**23,775**	23,716	23,616	**23,495**	23,495	23,492	**23,371**	**23,295**
48	21,147	20,382	**19,994**	20,107	19,920	**19,853**	19,994	19,736	**19,729**	**19,578**
56	18,513	18,237	**17,753**	17,575	17,003	**16,884**	17,265	16,905	**16,823**	**16,820**
64	17,430	17,300	**16,860**	15,560	15,399	**15,122**	15,186	15,052	**15,037**	**15,025**
*p*22810	$P_{max} = 8000$			$P_{max} = 10,000$			$P_{max} = 12,000$			Pmax = ∞
16	434,786	434,786	**433,430**	433,015	432,638	**431,966**	433,015	432,638	**431,966**	**425,113**
24	294,960	293,732	**292,548**	290,637	**288,933**	289,845	**288,633**	288,933	**288,633**	**284,851**
32	223,394	224,203	**222,487**	**218,322**	221,748	218,613	**218,322**	218,613	**218,322**	**214,934**
40	178,402	178,015	**177,763**	178,402	178,015	**177,763**	177,763	177,763	**177,312**	**175,946**
48	152,237	152,106	**151,001**	152,237	152,106	**151,001**	151,796	**150,413**	**150,413**	**147,756**
56	135,571	129,520	**129,292**	135,571	129,520	**129,292**	129,624	129,399	**129,292**	**126,570**
64	119,852	117,180	**116,625**	**116,625**	**116,625**	**116,625**	**116,625**	**116,625**	**116,625**	**116,625**
*p*93791	$P_{max} = 15,000$			$P_{max} = 20,000$			$P_{max} = 25,000$			Pmax = ∞
16	1,780,678	1,775,971	**1,771,797**	1,769,798	1,775,971	**1,769,673**	1,766,795	1,772,665	**1,763,891**	**1,720,725**
24	1,199,309	1,188,150	**1,178,986**	1,197,543	1,185,322	**1,178,986**	1,195,045	1,183,200	**1,178,741**	**1,150,467**
32	894,150	897,678	**888,064**	890,300	891,100	**888,064**	890,300	891,100	**888,038**	**864,460**
40	722,541	722,286	**718,005**	722,541	721,215	**710,508**	718,005	718,005	**710,508**	**695,200**
48	603,951	602,658	**592,200**	600,710	599,453	**592,200**	600,710	592,798	**592,200**	**579,761**
56	520,719	520,868	**513,658**	516,802	517,238	**512,188**	512,614	514,462	**509,565**	**496,559**
64	456,848	452,943	**450,523**	451,659	452,943	**449,633**	451,659	447,244	**447,244**	**435,838**
*p*34392	$P_{max} = 3700$			$P_{max} = 5000$			$P_{max} = 7000$			Pmax = ∞
16	1,058,834	1,032,630	**1,014,485**	999,487	993,478	**981,450**	993,721	992,627	**962,527**	**957,124**
24	876,014	876,014	**862,168**	862,168	792,553	**781,178**	702,852	698,818	**663,193**	**655,607**
32	858,762	858,762	**781,178**	792,553	792,553	**781,178**	544,579	544,579	**544,579**	**544,579**
40	838,643	792,553	**781,178**	792,553	792,553	**781,178**	544,579	544,579	**544,579**	**544,579**
48	792,553	792,553	**781,178**	792,553	792,553	**781,178**	544,579	544,579	**544,579**	**544,579**
56	792,553	792,553	**781,178**	792,553	792,553	**781,178**	544,579	544,579	**544,579**	**544,579**
64	785,910	785,910	**781,178**	785,910	781,178	**781,178**	544,579	544,579	**544,579**	544579
*g*1023	$P_{max} = 550$			$P_{max} = 800$			$P_{max} = 1000$			Pmax = ∞
16	33,597	33,491	**33,133**	31,730	31,560	**31,503**	31,431	31,209	**31,184**	**31,059**
24	29,400	29,338	**26,212**	23,586	22,149	**22,038**	21,909	21,445	**21,252**	**211,35**
32	27,584	27,527	**25,530**	20,733	20,733	**19,742**	17,165	17,212	**17,090**	**15,896**

Table 7 (*continued*)

W_{max}	GP	TP	WP	GP	TP	WP	GP	TP	WP	
40	27,584	27,526	**25,530**	20,733	20,733	**19,742**	16,945	16,945	**14,794**	**14,794**
48	27,584	27,526	**25,530**	20,733	20,733	**19,742**	15,444	15,299	**14,794**	**14,794**
56	27,584	27,526	**25,530**	20,733	20,733	**19,738**	15,444	15,299	**14,794**	**14,794**
64	27,584	27,526	**25,526**	20,733	20,733	**19,733**	15,444	15,299	**14,794**	**14,794**
*t*512505	$P_{max} = 21,000$			$P_{max} = 21,500$			$P_{max} = 22,000$			Pmax = ∞
16	10,531,175	10,531,171	**10,531,159**	10,531,171	10,531,171	**10,531,159**	10,531,048	10,530,995	**10,530,995**	**10,530,995**
24	10,531,633	1,453,565	**10,453,512**	10,454,163	10,453,565	**10,453,512**	10,454,163	10,454,163	**10,453,470**	**10,453,470**
32	5,359,882	5,326,372	**5,268,991**	5,268,875	5,268,875	**5,268,868**	5,268,936	5,269,411	**5,268,868**	**5,268,868**
40	5,269,239	5,228,759	**5,228,474**	5,228,827	5,228,759	**5,228,420**	5,228,420	5,228,420	**5,228,420**	**5,228,420**
48	5,269,139	5,269,139	**5,228,474**	5,228,474	5,228,474	**5,228,420**	5,228,420	5,228,420	**5,228,420**	**5,228,420**
56	5,269,139	5,269,139	**5,228,452**	5,228,474	5,228,474	**5,228,420**	5,228,420	5,228,420	**5,228,420**	**5,228,420**
64	5,269,139	5,269,139	**5,228,452**	5,228,474	5,228,474	**5,228,420**	5,228,420	5,228,420	**5,228,420**	**5,228,420**

Table 8
Comparisons of GP and TP with WP using PSO for different ITC'02 SoCs for different P_{max} and W_{max} (Contd.).

W_{max}	GP	TP	WP	GP	TP	WP	GP	TP	WP	
*q*12710	$P_{max} = 7000$			$P_{max} = 10,000$			$P_{max} = 15,000$			Pmax = ∞
16	5,032,534	4,895,391	**4,640,215**	3,443,783	3,483,203	**3,177,502**	**2,222,349**	**2,222,349**	**2,222,349**	**2,222,349**
24	5,032,534	4,895,391	**4,640,215**	3,443,783	3,483,203	**3,177,502**	**2,222,349**	**2,222,349**	**2,222,349**	**2,222,349**
32	5,032,534	4,895,391	**4,640,215**	3,443,783	3,483,203	**3,177,502**	**2,222,349**	**2,222,349**	**2,222,349**	**2,222,349**
40	5,032,534	4,895,391	**4,640,215**	3,443,783	3,483,203	**3,177,502**	**2,222,349**	**2,222,349**	**2,222,349**	**2,222,349**
48	5,032,534	4,895,391	**4,640,215**	3,443,783	3,483,203	**3,177,502**	**2,222,349**	**2,222,349**	**2,222,349**	**2,222,349**
56	5,032,534	4,895,391	**4,640,215**	3,443,783	3,483,203	**3,177,502**	**2,222,349**	**2,222,349**	**2,222,349**	**2,222,349**
64	4,895,391	4,675,639	**4,640,215**	3,443,783	3,483,203	**3,177,502**	**2,222,349**	**2,222,349**	**2,222,349**	**2,222,349**
*f*2126	$P_{max} = 5000$			$P_{max} = 7000$			$P_{max} = 10,000$			Pmax = ∞
16	613,237	496,251	**474,497**	492,447	492,447	**474,497**	**357,088**	**357,088**	**357,088**	**357,088**
24	613,237	496,251	**474,497**	492,447	492,447	**470,693**	357,088	357,088	**335,334**	**335,334**
32	613,237	496,251	**474,497**	492,447	492,447	**470,693**	357,088	357,088	**335,334**	**335,334**
40	613,237	496,251	**474,497**	492,447	492,447	**470,693**	357,088	357,088	**335,334**	**335,334**
8	613,237	496,251	**474,497**	492,447	492,447	**470,693**	357,088	357,088	**335,334**	**335,334**
56	613,237	496,251	**474,497**	492,447	492,447	**470,693**	357,088	357,088	**335,334**	**335,334**
64	613,237	496,251	**474,497**	492,447	492,447	**470,693**	357,088	357,088	**335,334**	**335,334**
*d*281	$P_{max} = 400$			$P_{max} = 500$			$P_{max} = 700$			Pmax = ∞
16	8718	8393	**8290**	8127	8127	**8051**	7906	7988	**7906**	**7906**
24	7214	7123	**6957**	6622	6622	**6532**	5567	5590	**5428**	**5428**
32	6379	6379	**6284**	5987	6049	**5987**	4163	4163	**3926**	**3926**
40	6125	6125	**5904**	5711	5749	**5512**	4163	4163	**3926**	**3926**
48	5808	5808	**5615**	5353	5353	**5353**	4163	4163	**3926**	**3926**
56	5693	5561	**5360**	5277	5277	**5194**	4163	4163	**3926**	**3926**
64	5596	5456	**5360**	5353	**5036**	**5036**	4163	4163	**3926**	**3926**
*u*226	$P_{max} = 700$			$P_{max} = 1000$			$P_{max} = 1200$			Pmax = ∞
16	**18,663**	**18,663**	**18,663**	**18,663**	**18,663**	**18,663**	**18,663**	**18,663**	**18,663**	**18,663**
24	**13,331**	**13,331**	**13,331**	**13,331**	**13,331**	**13,331**	**13,331**	**13,331**	**13,331**	**13,331**
32	**10,665**	**10,665**	**10,665**	**10,665**	**10,665**	**10,665**	**10,665**	**10,665**	**10,665**	**10,665**
40	**8084**	**8084**	**8084**	**8084**	**8084**	**8084**	**8084**	**8084**	**8084**	**8084**
48	**7999**	**7999**	**7999**	**7999**	**7999**	**7999**	**7999**	**7999**	**7999**	**7999**
56	**7999**	**7999**	**7999**	**7999**	**7999**	**7999**	**7999**	**7999**	**7999**	**7999**
64	**7999**	**7999**	**7999**	**7999**	**7999**	**7999**	**7999**	**7999**	**7999**	**7999**
*h*953	$P_{max} = 900$			$P_{max} = 1500$			$P_{max} = 2000$			Pmax = ∞
16	241,151	212,081	**211,533**	179,940	179,940	**177,496**	**119,357**	**119,357**	**119,357**	**119,357**
24	241,151	212,081	**211,533**	179,940	179,940	**177,496**	**119,357**	**119,357**	**119,357**	**119,357**
32	241,151	212,081	**211,533**	179,940	179,940	**177,496**	**119,357**	**119,357**	**119,357**	**119,357**
40	241,151	212,081	**211,533**	179,940	179,940	**177,496**	**119,357**	**119,357**	**119,357**	**119,357**
48	241,151	212,081	**211,533**	179,940	179,940	**177,496**	**119,357**	**119,357**	**119,357**	**119,357**
56	241,151	212,081	**211,533**	179,940	179,940	**177,496**	**119,357**	**119,357**	**119,357**	**119,357**
64	241,151	212,081	**211,533**	179,940	179,940	**177,496**	**119,357**	**119,357**	**119,357**	**119,357**
*a*586710	$P_{max} = 500$			$P_{max} = 750$			$P_{max} = 1000$			Pmax = ∞
16	42,355,169	**42,198,943**	**42,198,943**	42,117,536	42,117,536	**42,067,708**	**41,547,436**	**41,547,436**	**41,547,436**	**41,547,436**
24	28,797,909	28,450,601	**28,329,306**	28,329,306	28,316,901	**28,148,166**	27,907,180	**27,785,885**	**27,785,885**	**27,785,885**
32	22,973,201	22,498,601	**22,006,640**	21,694,396	21,694,396	**21,656,012**	21,343,768	21,343,768	**21,058,768**	**21,058,768**
40	**19,122,170**	**19,122,170**	**19,122,170**	**19,048,835**	**19,048,835**	**19,048,835**	**19,041,307**	**19,041,307**	**19,041,307**	**19,041,307**
48	**16,764,592**	**16,764,592**	**16,764,592**	15,315,467	15,315,467	**15,212,440**	15,112,162	15,112,162	**15,031,299**	**15,031,299**
56	**14,090,716**	**14,090,716**	**14,090,716**	14,669,018	14,669,018	**14,669,018**	13,481,897	**13,401,034**	**13,401,034**	**13,401,034**
64	13,401,034	13,401,034	**12,754,584**	12,754,584	12,700,205	**12,510,356**	11,567,464	11,567,464	**11,486,601**	**11,486,601**

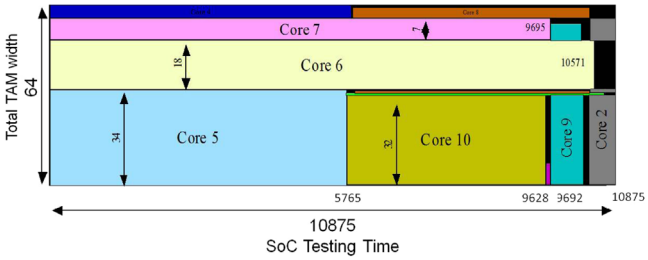

Fig. 11. Scheduling of *d*695 with P_{max}=2000 and W_{max}=64.

model are better than those for the simple global peak power model (GP) and the TAM-width dependent peak-power model (TP). The last columns of Tables 7 and 8 have reported the TAT results without considering power constraint. It may be noted that TAT reduces with relaxation of the power budget (P_{max}). Finally, it reaches its lowest value where power is not considered as a design constraint (i.e. $P_{max}=\infty$). In some of the cases, it reaches its minimal value at a higher P_{max} suggesting that no further improvement is possible with more relaxation of the power budget. Some of the SoCs show no improvement in TAT with an increase in the power budget or allocated test resources. This is due to the presence of bottleneck cores with much longer test time compared to other cores present in the SoC. This kind of cores dominates the schedule leaving no space of improvement by increasing P_{max} and W_{max} values.

To determine the CPU time saving achievable by our power model we have run the algorithm "*Schedule_Rectangle*" (Section 4.2.2) for a typical particle using three different power models global peak, window-based peak and cycle-accurate one. Experiments have been carried out on a system with the 2.40 GHz Intel Core2 Duo processor having 3 GB main memory. Table 5 notes the CPU times for the three approaches (single particle, single generation). It also reports the window-size that we have considered for each SoC. It can be noted that our window-based power model takes time one order of magnitude higher than the global peak based model. On the other hand, the cycle accurate power model takes time almost two orders of magnitude higher than the window-based model proposed in this paper. This justifies the usage of window-based power model for test scheduling approaches using evolutionary techniques, like PSO, to arrive at good scheduling results. Finally we have reported a resulting test schedule of window-based power model for *d*695 with P_{max} =2000 and W_{max}=64 in Fig. 11.

6. Conclusion

In this paper, a new window-based power model has been proposed to generate power-aware test schedule for SoC. The developed model has been shown to reduce the power over-estimation in the scheduling process. The model has been incorporated into a PSO based formulation to select test rectangles for cores and scheduling them using a 3-D bin packing approach. The power model can be extended to incorporate the features like voltage islands and multi-frequency testing. The PSO can be suitably modified to take care of the variances.

References

[1] E. Moghaddam, J. Rajski, S. Reddy, J. Janicki, Low test data volume low power at-speed delay tests using clock-gating, in: Test Symposium ATS, 2011 20th Asian, 2011, pp. 267–272. http://dx.doi.org/10.1109/ATS.2011.46.
[2] X. Chang, M. Zhang, G. Zhang, Z. Zhang, J. Wang, Adaptive clock gating technique for low power ip core in soc design, in: International Symposium on Circuits and Systems, 2007, ISCAS 2007, IEEE, 2007, pp. 2120–2123. http://dx.doi.org/10.1109/ISCAS.2007.378591.
[3] B. Yang, A. Sanghani, S. Sarangi, C. Liu, A clock-gating based capture power droop reduction methodology for at-speed scan testing, in: Design, Automation Test in Europe Conference Exhibition (DATE), 2011, 2011, pp. 1–7. http://dx.doi.org/10.1109/DATE.2011.5763042.
[4] L. Li, K. Choi, H. Nan, Effective algorithm for integrating clock gating and power gating to reduce dynamic and active leakage power simultaneously, in: 2011 12th International Symposium on Quality Electronic Design (ISQED), 2011, pp. 1–6. http://dx.doi.org/10.1109/ISQED.2011.5770706.
[5] E. Macii, L. Bolzani, A. Calimera, A. Macii, M. Poncino, Integrating clock gating and power gating for combined dynamic and leakage power optimization in digital cmos circuits, in: 2008 11th EUROMICRO Conference on Digital System Design Architectures, Methods and Tools, DSD '08, 2008, pp. 298–303. http://dx.doi.org/10.1109/DSD.2008.90.
[6] R.M. Chou, K.K. Saluja, V.D. Agrawal, Scheduling tests for vlsi systems under power constraints, IEEE Trans. Very Large Scale Integr. (VLSI) Syst. 5 (2) (1997) 175–185.
[7] Y. Huang, S.M. Reddy, W.-T. Cheng, P. Reuter, N. Mukherjee, C.-C. Tsai, O. Samman, Y. Zaidan, Optimal core wrapper width selection and soc test scheduling based on 3-d bin packing algorithm, in: 2002 Proceedings of International Test Conference, IEEE, 2002, pp. 74–82.
[8] J. Pouget, E. Larsson, Z. Peng, Soc test time minimization under multiple constraints, in: 2003 12th Asian Test Symposium, ATS 2003, IEEE, 2003, pp. 312–317.
[9] K. Chakrabarty, Design of system-on-a-chip test access architectures under place-and-route and power constraints, in: Proceedings of the 37th Annual Design Automation Conference, ACM, 2000, pp. 432–437.
[10] E. Larsson, Z. Peng, Test scheduling and scan-chain division under power constraint, in: 2001 Proceedings of 10th Asian Test Symposium, IEEE, 2001, pp. 259–264.
[11] C. Giri, S. Sarkar, S. Chattopadhyay, A genetic algorithm based heuristic technique for power constrained test scheduling in core-based socs, in: 2007 IFIP International Conference on Very Large Scale Integration, VLSI-SoC 2007, IEEE, 2007, pp. 320–323.
[12] X. Le Cui, X.M. Shi, H. Li, C.L. Lee, A shuffle frog-leaping algorithm for test scheduling of 2d/3d soc, in: 2012 IEEE 11th International Conference on Solid-State and Integrated Circuit Technology (ICSICT), IEEE, 2012, pp. 1–3.
[13] X.-l. Cui, W. Cheng, C.-r. Li, Effective soc test scheduling under peak power constraints based on the ant colony algorithm, Microelectron. Comput. 7 (2011) 001.
[14] S. Samii, E. Larsson, K. Chakrabarty, Z. Peng, Cycle-accurate test power modeling and its application to soc test scheduling, in: 2006 IEEE International Test Conference, ITC'06, IEEE, 2006, pp. 1–10.
[15] S. Samii, M. Selkala, E. Larsson, K. Chakrabarty, Z. Peng, Cycle-accurate test power modeling and its application to soc test architecture design and scheduling, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 27 (5) (2008) 973–977.
[16] J. Kennedy, R. Eberhart, Particle swarm optimization, in: 1995 Proceedings of IEEE International Conference on Neural Networks, vol. 4, 1995, pp. 1942–1948. http://dx.doi.org/10.1109/ICNN.1995.488968.
[17] E.J. Marinissen, V. Iyengar, K. Chakrabarty, Itc'02 Soc Test Benchmarks Web Site, 2006.
[18] E.J. Marinissen, V. Iyengar, K. Chakrabarty, A set of benchmarks for modular testing of socs, in: 2002 Proceedings of International Test Conference, IEEE, 2002, pp. 519–528.
[19] X. Kavousianos, K. Chakrabarty, A. Jain, R. Parekhji, Test schedule optimization for multicore socs: handling dynamic voltage scaling and multiple voltage islands, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 31 (11) (2012) 1754–1766.
[20] V. Sheshadri, V.D. Agrawal, P. Agrawal, Optimal power-constrained soc test schedules with customizable clock rates, in: 2012 IEEE International SOC Conference (SOCC), IEEE, 2012, pp. 271–276.
[21] S.K. Millican, K.K. Saluja, Formulating optimal test scheduling problem with dynamic voltage and frequency scaling, in: 2013 22nd Asian Test Symposium (ATS), IEEE, 2013, pp. 165–170.
[22] V. Sheshadri, V.D. Agrawal, P. Agrawal, Power-aware soc test optimization through dynamic voltage and frequency scaling, in: 2013 IFIP/IEEE 21st International Conference on Very Large Scale Integration (VLSI-SoC), IEEE, 2013, pp. 102–107.
[23] E. Marinissen, S. Goel, M. Lousberg, Wrapper design for embedded core test, in: 2000 Proceedings of International Test Conference, 2000, pp. 911–920. http://dx.doi.org/10.1109/TEST.2000.894302.
[24] V. Iyengar, K. Chakrabarty, E. Marinissen, Test wrapper and test access mechanism co-optimization for system-on-chip, in: 2001 Proceedings of International Test Conference, 2001, pp. 1023–1032. http://dx.doi.org/10.1109/TEST.2001.966728.
[25] A. Sehgal, S. Goel, E. Marinissen, K. Chakrabarty, Ieee p1500-compliant test wrapper design for hierarchical cores, in: 2004 Proceedings of International Test Conference, ITC 2004, 2004, pp. 1203–1212. http://dx.doi.org/10.1109/TEST.2004.1387393.
[26] V. Iyengar, K. Chakrabarty, E. Marinissen, Efficient wrapper/tam co-optimization for large socs, in: 2002 Proceedings on Design, Automation and Test in Europe Conference and Exhibition, 2002, pp. 491–498. http://dx.doi.org/10.1109/DATE.2002.998318.
[27] Y. Huang, W.-T. Cheng, C.-C. Tsai, N. Mukherjee, O. Samman, Y. Zaidan, S. Reddy, Resource allocation and test scheduling for concurrent test of core-based soc design, in: 2001 Proceedings of 10th Asian Test Symposium, 2001, pp. 265–270. http://dx.doi.org/10.1109/ATS.2001.990293.

[28] V. Iyengar, K. Chakrabarty, E. Marinissen, On using rectangle packing for soc wrapper/tam co-optimization, in: 2002 Proceedings of 20th IEEE VLSI Test Symposium, (VTS 2002), 2002, pp. 253–258. http://dx.doi.org/10.1109/VTS.2002.1011146.
[29] Y. Xia, M. Chrzanowska-Jeske, B. Wang, M. Jeske, Using a distributed rectangle bin-packing approach for core-based soc test scheduling with power constraints, in: 2003 International Conference on Computer Aided Design, ICCAD-2003, 2003, pp. 100–105. http://dx.doi.org/10.1109/ICCAD.2003.1257595.
[30] W. Zou, S. Reddy, I. Pomeranz, Y. Huang, Soc test scheduling using simulated annealing, in: 2003 Proceedings of 21st VLSI Test Symposium, 2003, pp. 325–330. http://dx.doi.org/10.1109/VTEST.2003.1197670.
[31] J.-Y. Wuu, T.-C. Chen, Y.-W. Chang, Soc test scheduling using the b*-tree based floorplanning technique, in: 2005 Proceedings of the Asia and South Pacific Design Automation Conference, ASP-DAC 2005, vol. 2, IEEE, 2005, pp. 1188–1191.
[32] Y. Yu, X. Peng, Y. Peng, A test scheduling algorithm based on two-stage ga, J. Phys.: Conf. Ser. 48 (2006) 658.
[33] J.-H. Ahn, S. Kang, Soc test scheduling algorithm using aco-based rectangle packing, Comput. Intell. 2006, 655–660.
[34] C. Giri, S. Sarkar, S. Chattopadhyay, Test scheduling for core-based socs using genetic algorithm based heuristic approach, in: Advanced Intelligent Computing Theories and Applications With Aspects of Artificial Intelligence, Springer, 2007, pp. 1032–1041.
[35] S. Goel, E. Marinissen, Effective and efficient test architecture design for socs, in: 2002 Proceedings of International Test Conference, 2002, pp. 529–538. http://dx.doi.org/10.1109/TEST.2002.1041803.
[36] S. Koranne, K. Chakrabarty, A novel reconfigurable wrapper for testing of embedded core-based socs and its associated scheduling algorithm, in: SOC (System-on-a-Chip) Testing for Plug and Play Test Automation, Springer, US, 2002, pp. 51–70.
[37] E. Larsson, H. Fujiwara, System-on-chip test scheduling with reconfigurable core wrappers, IEEE Trans. Very Large Scale Integr. (VLSI) Syst. 14 (3) (2006) 305–309.
[38] P.M. Rosinger, B.M. Al-Hashimi, N. Nicolici, Power profile manipulation: a new approach for reducing test application time under power constraints, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 21 (10) (2002) 1217–1225.
[39] P. Rosinger, B. Al-Hashimi, N. Nicolici, Power constrained test scheduling using power profile manipulation, iscas 2001.
[40] Synopsys Inc., TetraMax ATPG Guide, 2006.