A SIMULATED ANNEALING ALGORITHM FOR SYSTEM-ON-CHIP TEST SCHEDULING WITH POWER AND PRECEDENCE CONSTRAINTS*

HAIDAR M. HARMANANI and HASSAN A. SALAMY

Department of Computer Science and Mathematics
Lebanese American University, Byblos 1401, Lebanon

Revised 9 December 2006
Accepted 14 December 2006

This paper presents an efficient method to determine minimum system-on-chip (SOC) test schedules with precedence and power constraints based on simulated annealing. The problem is solved using a partitioned testing scheme with run to completion that minimizes the number of idle test slots. The method can handle SOC test scheduling with and without power constraints in addition to precedence constraints that preserve desirable orderings among tests. We present experimental results for various SOC examples that demonstrate the effectiveness of the method. The method achieved optimal test schedules in all attempted cases in a short CPU time.

Keywords: Embedded core testing; test scheduling; simulated annealing.

1. Introduction

Advances in modern VLSI technology allow to incorporate complete systems on a single chip using the system-on-chip (SOC) methodology through the use of pre-designed and pre-verified intellectual property (IP) cores. IP cores may be soft, firm, or hard and lead to a short design cycle by including processors, memories, buses, and interfaces on a chip. Integrating reusable cores into SOC involves complicated design and test issues since core-based designs are usually tested after assembly, at the end of the system implementation. Typically, a core is surrounded with test logic, known as test-wrapper, that serves as the interface between the core and the test access mechanism (TAM) and provides functions for a normal mode, an external test mode, and an internal test mode. During external test mode, the wrapper element drives the host chip in order to test the interconnect while during the internal test mode the wrapper element tests the core by observing the core output. Usually, a combination of built-in self-test (BIST) and external testing must be used to achieve high-fault coverage.

*This work was supported in part by a grant from the Lebanese National Council for Scientific Research (CNRS) and by the Lebanese American University.
One of the main challenges in core-based designs is test time reduction by maximizing the simultaneous test of all cores. The problem, known as test scheduling, determines the order in which various cores are tested and is equivalent, even for a simple SOC, to the NP-complete \textit{m-processor open shop scheduling} problem.\textsuperscript{2,7} One classical approach to solve the test scheduling problem is by organizing tests for the target cores or modules into so-called test sessions. A test session brings together the tests of compatible modules. This compatibility is checked with respect to the test resource sharing needs. Individual tests may be conflicting because:

1. they share common test resources such as a test bus or a test response compactor and
2. the power consumption during simultaneous testing exceeds the device power allowance.

Sessions-based test scheduling techniques assume either \textit{equal length test sessions} or \textit{unequal length test sessions}. During “equal length test session”, cores are arranged into sessions, where the length of each session is set to the longest test time in all sessions. In the “unequal length test session”, cores are arranged into sessions; however, the length of a specific session is the time taken to test the core requiring the longest time in that session. Recent techniques in test scheduling arrange cores for testing without sessions, where a test is initiated as soon as possible if the resource sharing and power constraints are not violated. These techniques, labeled as “sessionless”, partition testing with run to completion.\textsuperscript{6,14}

An effective test scheduling approach must minimize the test time while addressing resource conflicts among cores arising from the use of shared test access mechanisms, on-chip BIST engines, and power dissipation constraints. The average test time per SOC may be reduced further using an “abort at first fail” strategy. Thus, it is desirable to test components that are likely to fail first using precedence constraints that impose a partial ordering on the tests in a test suite.\textsuperscript{12} Similarly, since BIST is likely to detect more defects than external tests, it is desirable to apply BIST first to a core during manufacturing test. Finally, larger cores that are more likely to have defects are tested first. Embedding precedence constraints in the test schedule can play an important role in increasing the overall efficiency of a test suite.

Finally, power consumption complicates the testing of core-based systems as it impacts test parallelism. Power dissipation during testing is a function of time and depends on the switching activity resulting from the application of test vectors to the system.\textsuperscript{3} SOC in test mode can dissipate up to twice the amount of power it does in normal mode, since cores that do not normally operate in parallel may be tested concurrently in order to minimize testing time.\textsuperscript{18} Power-constrained test scheduling is therefore essential in order to limit the amount of concurrency during test application to ensure that the maximum power rating of the SOC is not exceeded.
1.1. Related work

There have been various reported approaches for the test scheduling problem in core-based systems. Craig et al.\textsuperscript{5} solved the general test scheduling problem by modeling tests compatibility using a test compatibility graph. A heuristic clique partitioning algorithm was used to generate suboptimal test schedules. Sugihara et al.\textsuperscript{23} formulated the test scheduling problem for core-based systems as a combinatorial optimization problem which is solved using a heuristic method. The authors made two restrictive assumptions: (1) Every core has its own BIST logic, and (2) external testing is carried out for one core at a time. Chakrabarty\textsuperscript{2} solved the test scheduling problem for core-based systems using an optimal formulation by mapping the problem to the \textit{m-processor open shop scheduling} problem, an NP-complete problem. The finish time of the schedule is mapped to the latest completion time of the individual processor schedule, while the length of the job is mapped to the test time. The problem is solved by minimizing the test finish time using a mixed integer linear programming (MILP) approach. For large instances where the MILP model is infeasible, the authors use a heuristic algorithm. Thus, although the objective of the ILP model was to derive optimal schedules, it was halted in various cases due to the problem’s complexity, resulting with nearly optimal solutions. The method was later extended by Iyengar et al.\textsuperscript{11} to include TAM optimization with core level wrapper optimizations. Larsson and Peng\textsuperscript{14,16} analyzed the test scheduling problem with power and test resource constraints, where an integrated SOC test framework is presented by analyzing the problem of test access mechanism design along with test scheduling. Flottes et al.\textsuperscript{6} presented a heuristic approach for test scheduling for SOC with power constraints. The advantage of the method is that it can handle large size problems in short time. Ravikumar et al.\textsuperscript{20} proposed a method to solve SOC test scheduling problem under power constraints while Harmanani and Salamy\textsuperscript{8} proposed a genetic algorithm to solve the power-constrained test scheduling problem. Other researchers tackled the SOC test scheduling problem in coordination with TAM assignment. For example, Huang et al.\textsuperscript{9} used a bin packing-based method to allocate test resources and to schedule test sets in order to achieve optimal concurrent SOC test. The objective is to minimize test application time for different TAMs under the constraint of peak power consumption. Su and Wu\textsuperscript{22} proposed a heuristic algorithm to solve the test scheduling problem while assigning TAM wires to the cores. Pouget et al.\textsuperscript{19} proposed a SOC test scheduling technique that minimizes test time while considering test powers limitations and test conflicts. Zhao and Upadhyaya\textsuperscript{25} proposed an efficient heuristic to solve the power-constrained test scheduling problem using the single-pair shortest path problem.

1.2. Problem description

This paper presents an efficient approach to SOC test scheduling based on test time, precedence, and power constraints. Given a set of cores, a set of test resources
consisting of TAMs and BIST engines, and a suite of tests such that each core requires at most one test set applied externally and one test set applied by a BIST engine, the problem we address in this paper is to minimize the overall test time by optimally determining the start times for the various cores in the test sets such that (i) no two tests for the same core are applied concurrently, (ii) there are no test resource conflicts, (iii) the schedule incorporates precedence constraints $i < j$, such that test $i$ is applied before test $j$, (iv) the BIST test for each core is applied before the external test for that core, and (v) the peak power during testing does not exceed a specified value, $P_{\text{max}}$. The method is motivated by the following:

- Test scheduling is an intractable problem that is necessary to reduce test time.
- Precedence constraints are important in order to increase the overall efficiency of a test suite by testing components that are most likely to fail first.

We assume that the test access architecture has been determined, and the cores have been assigned to test buses. No restrictions are placed either on the sharing of BIST logic among cores or on the use of multiple test buses for external testing.

The remainder of the paper is organized as follows. Section 2 introduces simulated annealing while Sec. 3 formulates the annealing SOC test scheduling problem and describes the cooling schedule, the neighborhood function, and the cost function. The annealing test scheduling algorithm is described in Sec. 4, while experimental results are presented in Sec. 5. Finally, we conclude with remarks in Sec. 6.

2. Simulated Annealing

Simulated annealing is a global stochastic method that is used to generate approximate solutions to very large combinatorial problems and was first introduced by Kirkpatrick and co-workers. The technique originates from the theory of statistical mechanics, based on the analogy between the annealing process of solids and the solving procedure for large combinatorial optimization. The annealing algorithm begins with an initial feasible configuration and proceeds to generate a neighboring solution by perturbing the current solution. If the cost of the neighboring solution is less than that of the current solution, the neighboring solution is accepted; otherwise, it is accepted or rejected with probability $p = e^{-\frac{\Delta C}{T}}$. The probability of accepting inferior solutions is a function of the temperature, $T$, and the change in cost between the neighboring solution and the current solution, $\Delta C$. The temperature is decreased during the optimization process, and thus the probability of accepting a worse solution decreases as well. The set of parameters controlling the initial temperature, stopping criterion, temperature decrement between successive stages, and the number of iterations for each temperature is called the cooling schedule. Typically, at the beginning of the algorithm, temperature $T$ is
large, and an inferior solution has a high probability of being accepted. During this period, the algorithm acts as a random search to find a promising region in the solution space. As the optimization process progresses, the temperature decreases, and there is a lower probability of accepting an inferior solution. The algorithm behaves like a down hill algorithm for finding the local optimum of the current region.

3. The Annealing Test Scheduling Formulation

The proposed test scheduling method starts with a compatibility graph of a set of cores and generates, through a sequence of transformations, a set of compact and optimal test schedules. The key elements in implementing the annealing test scheduling algorithm are (1) the definition of the initial configuration, (2) the definition of a neighborhood on the configuration space and a perturbation operator exploring it (3) the choice of the cost function, and (4) a cooling schedule. In what follows, we describe our annealing algorithm for SOC test scheduling with reference to Fig. 1, where a node $C_i(T_i, P_i)$ represents the test time and the power dissipation for core $i$.

3.1. Configuration representation

In order to solve the SOC test scheduling problem, we propose the configuration shown in Fig. 2(a). The representation is based on a vector where every cell corresponds to a core with a specific test start time, $S_i$. The core test finish time, $F_i$, is

![Fig. 1. Simple SOC compatibility graph.](image)
equal to the test start time plus the core test time, $T_i$, that is, $F_i = T_i + S_i$. Note that the test start time, $S_i$, is not constant, and it changes to the end times of other cores as the algorithm explores the neighborhood of the solution. Figure 2(b) shows a sample configuration using the compatibility graph in Fig. 1. The corresponding test schedule is shown in Fig. 2(c).

### 3.2. Initial configuration

The initial configuration is chosen to be a random serial schedule, which is the longest possible test schedule. However, when considering precedence constraints, a completely random initial configuration is not feasible as it does not meet the precedence constraints. Thus, the initial configuration in this case is chosen based on the topological sorting of all tests represented in the precedence constraints graph. Topological sorting is an ordering of the nodes of a directed acyclic graph such that for every directed path from node $u$ to node $v$, $u$ appears before $v$ in the topological order. The result of the topological sorting is a valid serial schedule that we use as an initial configuration when precedence constraints are incorporated.

### 3.3. Neighborhood transformation

A neighboring solution is selected by randomly selecting a core $i$ from the current configuration and changing its starting time $S_i$ to the end time $F_j$ of a randomly chosen core $j$ ($i \neq j$), where $0 \leq i, j \leq N_c + 1$. If core $j$ is selected such that $j = N_c + 1$, then the start time of core $i$ is set to 0. The neighborhood solution is followed with a deterministic compaction transformation, Fill Gap, that compacts
the solution by filling the gaps or the idle slots among the scheduled cores. The algorithm for the Fill Gap transformation is shown in Fig. 3.

We illustrate the neighborhood transformation using the example in Fig. 4(a). For instance, a new neighborhood solution is selected by randomly selecting core \( C_3 \) and changing its test start time to the test end time of another randomly selected core, \( C_2 \). The resulting configuration schedules core \( C_3 \) at \( t = 125 \). The test schedule for the resulting configuration is shown in Fig. 4(b).

---

**Fig. 3.** Fill gap transformation pseudo-code.

```pseudo
Fill_Gap(Configuration, i))
{
    Core\((t_i, f_i) \leftarrow \text{Configuration} [i]\)
    List_1 = All cores whose start_time > \( t_i \)
    List_2 = All cores such that start_time \( \leq \) \( t_i \) < end_time
    if List_2 is empty
        subtract \( f_i - t_i \) from all nodes in List_1
    else {
        max_1 = Largest end_time in List_2
        for (all nodes \((t_j, f_j) \in \text{List}_1)\) {
            max_2 = Largest end_times for cores in List_2 \( \leq \) \( t_j \)
            if \( t_j < \text{max}_1 \)
                if \( (f_i - \text{max}_2 > 0) \)
                    \( t_j = t_j - (f_i - \text{max}_2) \)
                else
                    \( t_j = t_j - \text{max}_3 \)
            else
                max_3 = Largest end_time in List_1 and List_2 \( \leq \) \( t_j \)
                \( t_j = t_j - \text{max}_3 \)
            }
        }
    }
}
```

---

**Fig. 4.** (a) Neighborhood transformation, and (b) test schedule.
3.4. Cost function

Given a set of $N_c$ cores $\{C_1, C_2, \ldots, C_{N_c}\}$ with corresponding test times $\{T_1, T_2, \ldots, T_{N_c}\}$ and corresponding peak powers $\{P_1, P_2, \ldots, P_{N_c}\}$, and a partial ordering on the test suite, the objective function is to minimize the overall test time by optimally determining the start times for the various cores in the test sets such that the peak power is not exceeded during testing and the specified ordering in the test suite is preserved. The peak power dissipation is estimated as $\sum_C P_i$.

3.5. Cooling schedule

The cooling schedule is the set of parameters controlling the initial temperature, the stopping criterion, the temperature decrement between successive stages, and the number of iterations for each temperature. The cooling schedule was empirically determined as follows:

1. $T_{\text{init}} = 4000$.
2. The temperature reduction multiplier, $\alpha$, is set to 0.99.
3. The number of iterations, $M$, is set to 5, while the iteration multiplier, $\beta$, is set to be 1.05.

The algorithm stops when the current temperature, $T$, is below 0.001.

4. Test Scheduling Algorithm

Each configuration represents an intermediate test schedule that has a different cost. During every annealing iteration, the neighborhood of the configuration is explored. The algorithm must ensure the following:

1. Cores that are tested concurrently are compatible.
2. The test suite ordering, as specified in the precedence graph, is respected.
3. The peak power of cores that are tested concurrently does not exceed the maximum power rating of the SOC, $P_{\text{max}}$.

Two test sets are conflicting if (i) they share resources such as an external bus; (ii) they share BIST test set for cores that share a BIST resource or they are the external and BIST components of a core’s test set. Cores compatibility is represented using a compatibility graph where nodes represent tests while edges indicate compatibility among nodes. The precedence constraints are represented using a directed graph where a directed edge between nodes $i$ and $j$ indicates that node $j$ cannot start unless node $i$ is completely finished. Thus, $S_j \geq S_i + T_i$, where $S_i$ is the test start time of core $i$ and $T_i$ is its test time duration.

The algorithm, shown in Fig. 5, starts by selecting an initial configuration and then a sequence of iterations is performed. In each iteration a new configuration is
A Simulated Annealing Algorithm for SOC Test Scheduling

```plaintext
Annealing_TestScheduling()
{
  S₀ = Initial solution
  α = Cooling rate
  β = Iteration multiplier
  T₀ = Initial temperature
  MaxTime = Total allowed time for the annealing process
  M₀ = Time until next parameter update
  BestS = Best solution
  T = T₀
  CurrentS = S₀
  CurrentCost = Cost(CurrentS)
  BestCost = Cost(BestS)
  Time = 0
  do {
    M = M₀
    do {
      NewS = Neighbor(CurrentS);
      NewCost=Cost(NewS)
      ∆Cost = NewCost - CurrentCost
      If (∆Cost < 0)
        CurrentS=NewS
        CurrentCost=Cost(CurrentS);
      If (NewCost < BestCost) then
        BestS=NewS
        BestCost = Cost(BestS)
      else if (Random < e⁻∆Cost/T) then
        CurrentS=NewS
        CurrentCost = Cost(CurrentS);
      M = M - 1
    } while (M ≥ 0)
    Time = Time + M₀;
    T = α * T;
    M₀ = β * M₀;
  } while (Time > MaxTime and T > 0.001); Return(BestS);
}
```

Fig. 5. Annealing SOC test scheduling algorithm.

generated in the neighborhood of the original configuration by randomly changing the test start time of a randomly selected core to the finish time of another randomly chosen core. The solution is always feasible as the neighborhood transformation uses a constructive approach that generates feasible test schedules. The variation in the cost functions, ∆C, is computed, and if negative then the transition from Ci to Ci+1 is accepted. If the cost function increases, the transition is accepted with a probability based on the Boltzmann distribution. The temperature is gradually decreased throughout the algorithm from a sufficiently high starting value, Tinit = 4000, where almost every proposed transition, positive or negative, is accepted to a freezing temperature, Tf = 0.001, where no further changes occur.
5. Experimental Results

5.1. Benchmarks

The proposed algorithm was implemented using the Java language on a Pentium Centrino with 2.13 GHz clock and 1 GB of RAM. The algorithm was tested on various benchmark examples from the literature. We compare the proposed approach with the techniques proposed by Chou et al.,4 Flottes et al.,6 Larsson et al.,15,17 Muresan et al.,18 and Zhao and Upadhyaya.24 Detailed results comparisons are shown in Tables 5 and 6.

5.1.1. Muresan SOC

The Muresan SOC example was reported by Muresan et al.18 and used later by other researchers for comparison purposes. The SOC, shown in Table 1, has 10 tests and a power constraint of $P_{\text{max}} = 12$. The example was later modified to include 20 tests with a power constraint of $P_{\text{max}} = 15$.

The test schedule proposed by Muresan et al.18 for the Muresan 10 SOC leads to a total testing time of 29 cycles. We generate the schedule shown in Fig. 6 with a total testing time of 25 cycles. The example was test scheduled in 0.284 CPU seconds. The second SOC, Muresan 20, was test scheduled by Larsson et al.13,14 leading to 49 and 47 cycles, respectively. Our algorithm generated a test schedule with a 39 cycles total test time, an improvement of 20.5% and 25.6%, respectively. The Muresan 20 SOC test schedule was generated in 0.324 CPU seconds.

5.1.2. Muresan II SOC

The Muresan II SOC, whose characteristics are shown in Table 2, was first reported by Muresan et al.18 and used later by Flottes et al.6 for comparison purposes. The example was test scheduled in 31,000,000 cycles. The authors improved the “unequal length session approach” by allowing several cores to be tested sequentially within a session. The example was test scheduled based on our annealing approach using a “sessionless” scheme in 23,000,000 cycles, which is the optimal test schedule (Fig. 7). The example was test scheduled in 0.475 CPU seconds.

<table>
<thead>
<tr>
<th>Table 1. Test data for the Muresan SOC.</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_1(9,9,t_2,t_3,t_5,t_6,t_8,t_9)$</td>
</tr>
<tr>
<td>$t_2(4,8,t_1,t_3,t_7,t_8)$</td>
</tr>
<tr>
<td>$t_3(1,8,t_1,t_2,t_4,t_7,t_9,t_{10})$</td>
</tr>
<tr>
<td>$t_4(6,6,t_3,t_5,t_7,t_8)$</td>
</tr>
<tr>
<td>$t_5(5,5,t_1,t_4,t_9,t_{10})$</td>
</tr>
</tbody>
</table>
A Simulated Annealing Algorithm for SOC Test Scheduling  

5.1.3. Flottes SOC

The Flottes SOC example was first reported in Flottes et al. The example includes 14 cores with a power constraint of $P_{\text{max}} = 30$. The testing time obtained in Ref. 6 for this example is 52,000 cycles while our method leads to the schedule shown in
Fig. 8 with a total testing time of 46,000 cycles. The test schedule was generated in 2.031 CPU seconds.

5.1.4. ASIC Z

The ASIC Z example, whose details are shown in Table 3, was reported by Zorian.\textsuperscript{27} The example consists of four RAMs, two ROMs, and three random logic blocks. Our proposed approach results in a test schedule with a test time of 262 cycles for a maximum power constraint, $P_{\text{max}}$, of 900 mW. The schedule, shown in Fig. 9, was generated in 0.6 CPU seconds.

Table 3. Test length and power data for ASIC Z system.

<table>
<thead>
<tr>
<th>Block</th>
<th>Power (mW)</th>
<th>Test length</th>
</tr>
</thead>
<tbody>
<tr>
<td>RL1</td>
<td>295</td>
<td>134</td>
</tr>
<tr>
<td>RL2</td>
<td>352</td>
<td>160</td>
</tr>
<tr>
<td>RF</td>
<td>95</td>
<td>10</td>
</tr>
<tr>
<td>RAM1</td>
<td>282</td>
<td>69</td>
</tr>
<tr>
<td>RAM2</td>
<td>241</td>
<td>61</td>
</tr>
<tr>
<td>RAM3</td>
<td>213</td>
<td>38</td>
</tr>
<tr>
<td>RAM4</td>
<td>96</td>
<td>23</td>
</tr>
<tr>
<td>ROM1</td>
<td>279</td>
<td>102</td>
</tr>
<tr>
<td>ROM2</td>
<td>279</td>
<td>102</td>
</tr>
</tbody>
</table>

Fig. 9. Power-constrained test schedule for ASIC Z.
5.1.5. Ericsson design

The Ericsson SOC first reported by Larsson et al.\textsuperscript{17} consists of 18 cores including eight DSPs, a DSP control, two memory banks, a common program memory and a common data memory, a control unit for each memory bank, common data memory controller and common program memory controller, and five logic blocks. The Ericsson SOC has a total of 170 tests with a maximal allowed power consumption of \( P_{\text{max}} = 5125 \text{ mW} \). The Ericsson SOC was scheduled using our method in 30,899 cycles, which is the optimal in this case.

5.1.6. Zhao SOC

The Zhao SOC example is a hypothetical but a non-trivial SOC that consists of seven embedded cores from the ISCAS’85 and ISCAS’89 combinational and sequential benchmarks. The example was first reported in Ref. 24 and then used later by Zhao and Upadhyaya\textsuperscript{26} for comparison purposes. Each core is provided with a test set that includes, among others, the number of test patterns, number of capacitance nodes, and the peak switching frequency. The power constraint, \( P_{\text{max}} \), is assumed to be 900 mW. The compatibility graph for this example is shown in Fig. 10, where each node \( T_i(\text{test time, power}) \) represents the test time and the power dissipation of core \( i \).

The example was scheduled in 618,915 cycles using the block-test method\textsuperscript{4} and in 561,672 cycles using the greedy best-fit algorithm.\textsuperscript{15} Both methods were
implemented by Zhao and Upadhyaya\textsuperscript{24} who scheduled the example in 493,394 cycles. The example was test scheduled using our proposed method in 493,394 cycles, as shown in Fig. 11, a significant improvement of 13.83\% over Ref. 15 and 25.44\% over Ref. 4. The example was test scheduled in 0.840 CPU seconds.

5.1.7. $d5018$ SOC

The $d5018$ system, whose details are shown in Table 4, is an example SOC that consists of eight ISCAS benchmark cores. The example was scheduled by Chakrabarty\textsuperscript{2} using the shortest-task-first and later in Ref. 10 using precedence constraints. While the total testing time obtained using the shortest-task-first is 7851 cycles, the total testing time obtained using our approach is 6809 cycles, an improvement of 15.30\%.

The schedule for the $d5018$ example obtained by our approach is shown in Fig. 12 and was test scheduled in 1.114 CPU seconds.

The $d5018$ SOC example is next test scheduled by considering power constraints. Iyengar and Chakrabarty\textsuperscript{10} test scheduled this example using an MILP formulation in 7985 cycles with a power constraint of $P_{\text{max}} = 950$ mW. The power-constrained test schedule for $d5018$ using our approach is shown in Fig. 13, where the total testing time is 6809 cycles, an improvement of 17.27\%. The example was test scheduled in 1.785 CPU seconds.

The example was next test scheduled by considering power and precedence constraints using a test architecture that consists of one external test bus and four BIST engines and assuming that for each core the BIST test should precede the external test. The total test time obtained using our approach is 7065 cycles. The schedule for the example is shown in Fig. 14 and was generated in 4 CPU seconds.

Finally, the $d5018$ example was test scheduled by adding precedence constraints to the 950 mW power constraint resulting with the schedule shown in Fig. 15. Our method test scheduled the example in 7065 cycles, a substantial improvement of 23.47\% over the 8723 cycles test time reported in Ref. 10. The example was generated in 4.5 CPU seconds.
Table 4. Test scheduling results with power considerations.

<table>
<thead>
<tr>
<th>Design</th>
<th>Number of Cores</th>
<th>Approach</th>
<th>Test Time</th>
<th>$P_{\text{max}}$</th>
<th>Difference to Optimum (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Muresan 10$^{18}$</td>
<td>10</td>
<td>Optimal</td>
<td>25</td>
<td>12</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Muresan et al.$^{18}$</td>
<td>29</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Larsson and Peng$^{15}$</td>
<td>26–28</td>
<td>4–12</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Flottes et al.$^{6}$</td>
<td>25</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Proposed approach</td>
<td>25</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Muresan 20$^{18}$</td>
<td>10</td>
<td>Optimal</td>
<td>39</td>
<td>15</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TSP$^{13}$</td>
<td>47</td>
<td>20.5</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Muresan$^{15}$</td>
<td>49</td>
<td>25.6</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Proposed approach</td>
<td>39</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Muresan II$^{18}$</td>
<td>9</td>
<td>Optimal</td>
<td>23,000,000</td>
<td>12</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Muresan et al.$^{18}$</td>
<td>31,000,000</td>
<td>34.78</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Flottes et al.$^{6}$</td>
<td>23,000,000</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Proposed approach</td>
<td>23,000,000</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Flottes$^{6}$</td>
<td>14</td>
<td>Optimal</td>
<td>46,000</td>
<td>30</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Flottes et al.$^{6}$</td>
<td>52,000</td>
<td>13</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Proposed approach</td>
<td>46,000</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>ASIC Z$^{27}$</td>
<td>9</td>
<td>Optimal</td>
<td>262</td>
<td>900</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Larsson and Peng$^{15}$</td>
<td>262</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Proposed approach</td>
<td>262</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Ericsson$^{17}$</td>
<td>18</td>
<td>Optimal</td>
<td>30,899</td>
<td>5125</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Larsson et al.$^{14,17}$</td>
<td>30,899–34,762</td>
<td>0–12.5</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Proposed approach</td>
<td>30,899</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Zhao SOC$^{24}$</td>
<td>7</td>
<td>Optimal</td>
<td>493,394</td>
<td>900</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Chou et al.$^{4}$</td>
<td>618,915</td>
<td>25.44</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Larsson and Peng$^{15}$</td>
<td>561,672</td>
<td>13.83</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Zhao and Upadhyaya$^{24}$</td>
<td>493,394</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Proposed approach</td>
<td>493,394</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>
5.2. Results analysis

From the various attempted SOC benchmarks, we have noted the following observations:

- Our method performs consistently well for all SOC examples and obtains the optimal test scheduling time, irrespective of the number of tests or the number of constraints. For example, while some methods obtain optimal schedules for some SOCs they fail to do so for other SOCs. This can be best illustrated in Table 5. For example, Flottes et al.⁶ achieve optimal schedules for the Muresan
A Simulated Annealing Algorithm for SOC Test Scheduling

Fig. 14. Test schedule for d5018 under precedence constraints on BIST and external tests.

Fig. 15. Test schedule for d5018 using power and precedence constraints.
Table 5. Test data for the d5018 system.

<table>
<thead>
<tr>
<th>Core</th>
<th>BIST Test Time</th>
<th>External Test Time</th>
<th>Power in BIST Mode (mW)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>256</td>
<td>134</td>
<td>54</td>
</tr>
<tr>
<td>2</td>
<td>2048</td>
<td>2543</td>
<td>159</td>
</tr>
<tr>
<td>3</td>
<td>2048</td>
<td>1357</td>
<td>453</td>
</tr>
<tr>
<td>4</td>
<td>256</td>
<td>454</td>
<td>57</td>
</tr>
<tr>
<td>5</td>
<td>256</td>
<td>1903</td>
<td>324</td>
</tr>
<tr>
<td>6</td>
<td>256</td>
<td>242</td>
<td>72</td>
</tr>
<tr>
<td>7</td>
<td>2048</td>
<td>—</td>
<td>792</td>
</tr>
<tr>
<td>8</td>
<td>1024</td>
<td>176</td>
<td>75</td>
</tr>
</tbody>
</table>

Table 6. Test scheduling results for the d5018 SOC.

<table>
<thead>
<tr>
<th>Scheduling constraints</th>
<th>Approach</th>
<th>Test Time</th>
<th>Difference to Optimum (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resource</td>
<td>Optimal</td>
<td>6809</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td>Chakrabarty²</td>
<td>7851</td>
<td>15.30</td>
</tr>
<tr>
<td></td>
<td>Proposed approach</td>
<td>6809</td>
<td>0</td>
</tr>
<tr>
<td>Power (950 mW)</td>
<td>Optimal</td>
<td>6809</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td>Chakrabarty²</td>
<td>7851</td>
<td>15.30</td>
</tr>
<tr>
<td></td>
<td>Iyengar and Chakrabarty¹⁰</td>
<td>7985</td>
<td>17.27</td>
</tr>
<tr>
<td></td>
<td>Flottes et al.⁶</td>
<td>6809</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Proposed approach</td>
<td>6809</td>
<td>0</td>
</tr>
<tr>
<td>Precedence on BIST and</td>
<td>Optimal</td>
<td>7065</td>
<td>—</td>
</tr>
<tr>
<td>external test constraints</td>
<td>Chakrabarty²</td>
<td>7065</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Proposed approach</td>
<td>7065</td>
<td>0</td>
</tr>
<tr>
<td>Power, precedence, and</td>
<td>Optimal</td>
<td>7065</td>
<td>—</td>
</tr>
<tr>
<td>test constraints</td>
<td>Chakrabarty²</td>
<td>8723</td>
<td>23.47</td>
</tr>
<tr>
<td></td>
<td>Proposed approach</td>
<td>7065</td>
<td>0</td>
</tr>
</tbody>
</table>

10, but fail to do so for the Flottes SOC.⁶ The same observation can be made in Table 6 where Chakrabarty² obtains the optimal test schedule for precedence on BIST and external test constraints case and sub-optimal test schedules in the remaining three cases.

- As indicated earlier, the test scheduling problem is intractable. Most reported techniques are based on deterministic algorithms whose complexity is related to the number of tests. Our non-deterministic algorithm performs well due to two main reasons. The first is the configuration encoding and the efficient neighborhood transformation that allow the system to locally explore compact test schedules. The second reason is that we combine our method with a deterministic transformation that reduces idle slots in the schedule thus drastically improving the convergence time.

6. Conclusion

We have presented a fast and efficient method for the test scheduling problem of core-based systems using a partitioned testing scheme with run to completion. The
method is based on a simulated annealing algorithm that explores the decision space in a very short time. The method minimizes the overall test application time of a SOC through efficient and compact test schedules and handles SOC test scheduling with and without power constraints in addition to precedence constraints. We presented experimental results for various SOC examples that demonstrate the effectiveness of our method. The method achieved optimal test schedules in all attempted cases.

References