International Journal of Computational Intelligence and Applications
Vol. 6, No. 4 (2006) 511–530

Imperial College Press
www.icpress.co.uk

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 2010, 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 predesigned 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.[1] 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 *m-processor open shop scheduling* problem.[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 *equal length test sessions* or *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.[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.[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.[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.[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.*[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.*[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[2] solved the test scheduling problem for core-based systems using an optimal formulation by mapping the problem to the *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.*[11] to include TAM optimization with core level wrapper optimizations. Larsson and Peng[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.*[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.*[20] proposed a method to solve SOC test scheduling problem under power constraints while Harmanani and Salamy[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.*[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[22] proposed a heuristic algorithm to solve the test scheduling problem while assigning TAM wires to the cores. Pouget *et al.*[19] proposed a SOC test scheduling technique that minimizes test time while considering test powers limitations and test conflicts. Zhao and Upadhyaya[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, $P_{\max}$. The method is motivated by the following:

- Test scheduling is an *intractable* problem that is necessary to reduce test time. This work presents an efficient and fast simulated annealing algorithm to solve the above problem. The proposed method obtains optimal test schedules with precedence and power constraints for large SOCs.
- 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.[21] 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, Δ_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

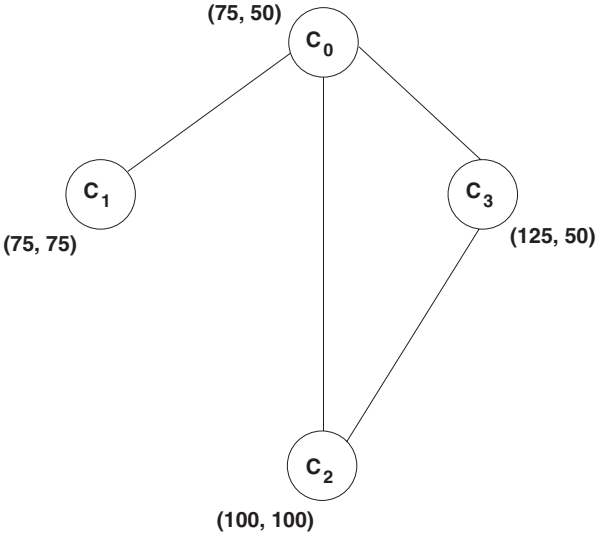

Fig. 1. Simple SOC compatibility graph.

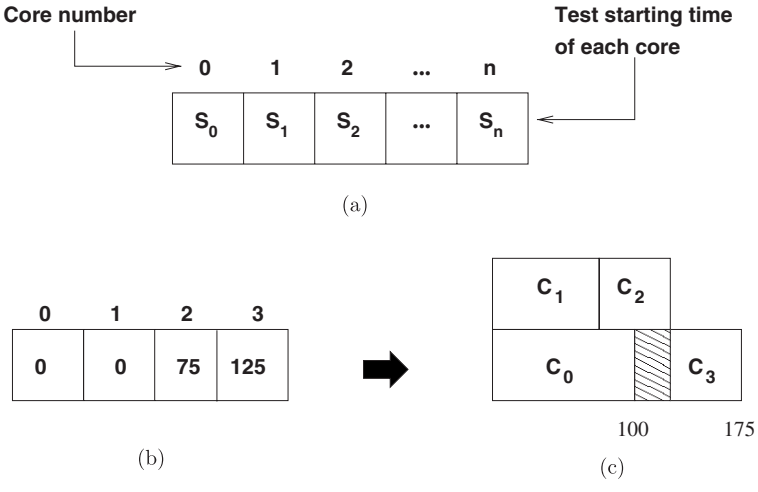

Fig. 2. (a) Configuration representation, (b) sample configuration, and (c) corresponding test schedule.

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 $\neq i$ 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).


```
Fill_Gap(Configuration, i))
{
  Core(t_i, f_i) ← Configuration [i]
  List_1 = All cores whose start_time > t_i
  List_2 = All cores such that start_time ≤ 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 u(t_j, f_j) ∈ List_1) {
        max_2 = Largest end_times for cores in List_2 ≤ t_j
        if (t_j < max_1)
          if (f_i − max_2 > 0)
            t_j = t_j − (f_i − max_2)
        else
          if t_j > max_1)
            if (f_i − max_1 > 0)
              t_j = t_j − (f_i − max_1)
            else
              max_3 = Largest end_time in List_1 and List_2 ≤ t_j
              t_j = t_j − max_3
    }
  }
}
```

Fig. 3. Fill gap transformation pseudo-code.

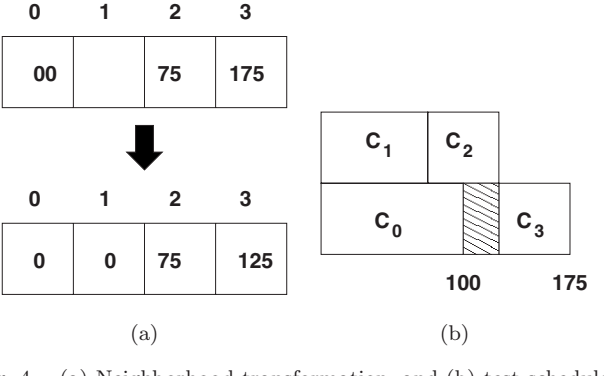

Fig. 4. (a) Neighborhood transformation, and (b) test schedule.

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 $\sum_{C_i} P_i$.

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, α, is set to 0.99.
3. The number of iterations, M, is set to 5, while the iteration multiplier, β, 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, $P_{\max}$.

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

```
Annealing_TestScheduling()
{
    S_0 = Initial solution
    α = Cooling rate
    β = Iteration multiplier
    T_0 = Initial temperature
    MaxTime = Total allowed time for the annealing process
    M_0 = Time until next parameter update
    BestS = Best solution
    T = T_0
    CurrentS = S_0
    CurrentCost = Cost(CurrentS)
    BestCost = Cost(BestS)
    Time = 0
    do {
        M = M_0
        do {
            NewS = Neighbor(CurrentS);
            NewCost=Cost(NewS)
            Δ_Cost = NewCost − CurrentCost
            If (Δ_Cost < 0)
                CurrentS=NewS
                CurrentCost=Cost(CurrentS);
                If (NewCost < BestCost) then
                    BestS=NewS
                    BestCost = Cost(BestS)
            else if (Random < e^(−Δ_Cost/T) then
                CurrentS=NewS
                CurrentCost = Cost(CurrentS);
            M = M − 1
        } while (M ≥ 0)
        Time = Time + M_0;
        T = α ∗T;
        M_0 = β ∗ M_0;
    } 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, Δ_C, is computed, and if negative then the transition from C_i to C_{i+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, $T_{\text{init}} = 4000$, where almost every proposed transition, positive or negative, is accepted to a freezing temperature, $T_{\text{f}} = 0.001$, 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 $P_{\max} = 12$. The example was later modified to include 20 tests with a power constraint of $P_{\max} = 15$.

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 1. Test data for the Muresan SOC.

$t_1(9, 9, t_2, t_3, t_5, t_6, t_8, t_9)$	$t_6(2, 4, t_1, t_7, t_8, t_9)$
$t_2(4, 8, t_1, t_3, t_7, t_8)$	$t_7(1, 3, t_2, t_3, t_4, t_6, t_8, t_9)$
$t_3(1, 8, t_1, t_2, t_4, t_7, t_9, t_{10})$	$t_8(4, 2, t_1, t_2, t_4, t_6, t_7, t_9, t_{10})$
$t_4, (6, 6, t_3, t_5, t_7, t_8)$	$t_9(12, 1, t_1, t_3, t_5, t_6, t_7, t_8, t_{10})$
$t_5(5, 5, t_1, t_4, t_9, t_{10})$	$t_{10}(7, 1, t_3, t_5, t_8, t_9)$

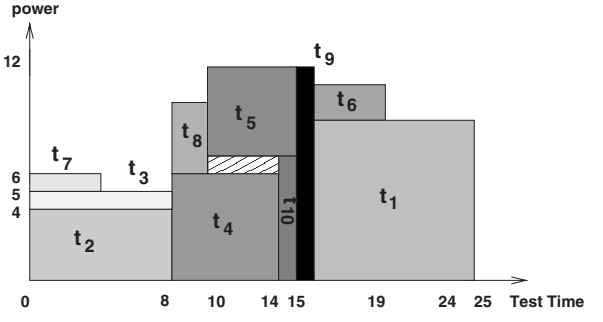

Fig. 6. Test schedule using a sessionless scheme for the Muresan I SOC example.[18]

Table 2. Test data for the Muresan II SOC example.[6]

Core	P_i	D_i	Share test with
1	6	16,000	6 7 8
2	5	10,000	4 5 6 7
3	4	9000	6 7 9
4	2	7000	5
5	8	4000	2 4 7
6	2	3000	1 2 3
7	2	2000	1 2 3 5
8	2	1000	1
9	1	3000	3

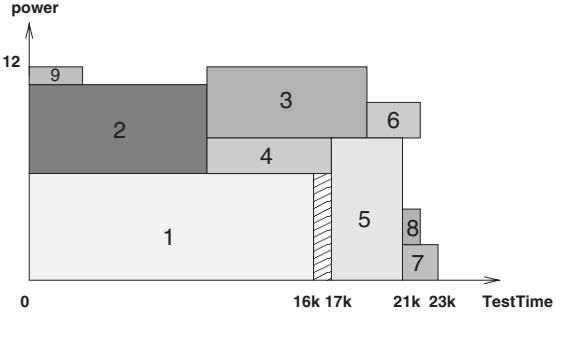

Fig. 7. Power-constrained test schedule for the Muresan II SOC example.[6]

5.1.3. *Flottes SOC*

The *Flottes* SOC example was first reported in Flottes *et al.*[6] The example includes 14 cores with a power constraint of $P_{\max} = 30$. The testing time obtained in Ref. 6 for this example is 52,000 cycles while our method leads to the schedule shown in

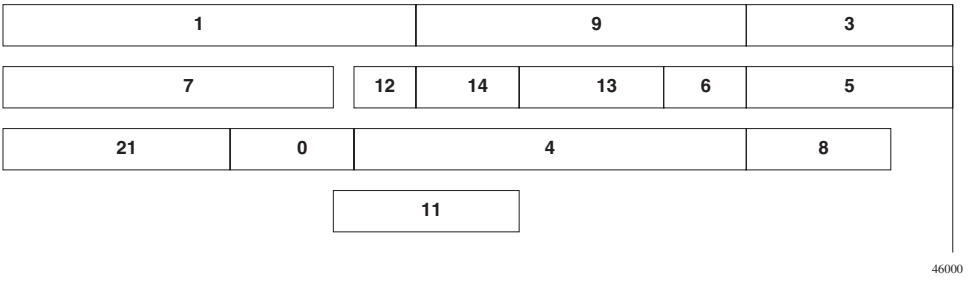

Fig. 8. Test schedule for Flottes SOC example.[6]

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.[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, $P_{\max}$, 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.

Block	Power (mW)	Test length
RL1	295	134
RL2	352	160
RF	95	10
RAM1	282	69
RAM2	241	61
RAM3	213	38
RAM4	96	23
ROM1	279	102
ROM2	279	102

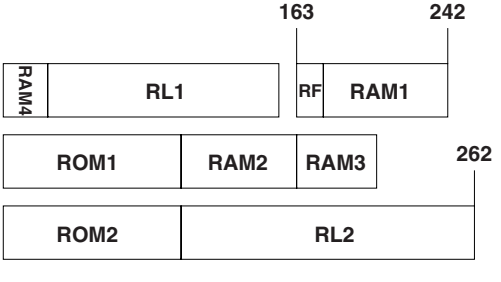

Fig. 9. Power-constrained test schedule for ASIC Z.

5.1.5. *Ericsson design*

The Ericsson SOC first reported by Larsson *et al.*[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 $P_{\max} = 5125\,\text{mW}$. 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[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, $P_{\max}$, is assumed to be 900 mW. The compatability graph for this example is shown in Fig. 10, where each node $T_i(test\_time, power)$ represents the test time and the power dissipation of core i.

The example was scheduled in 618,915 cycles using the block-test method[4] and in 561,672 cycles using the greedy best-fit algorithm.[15] Both methods were

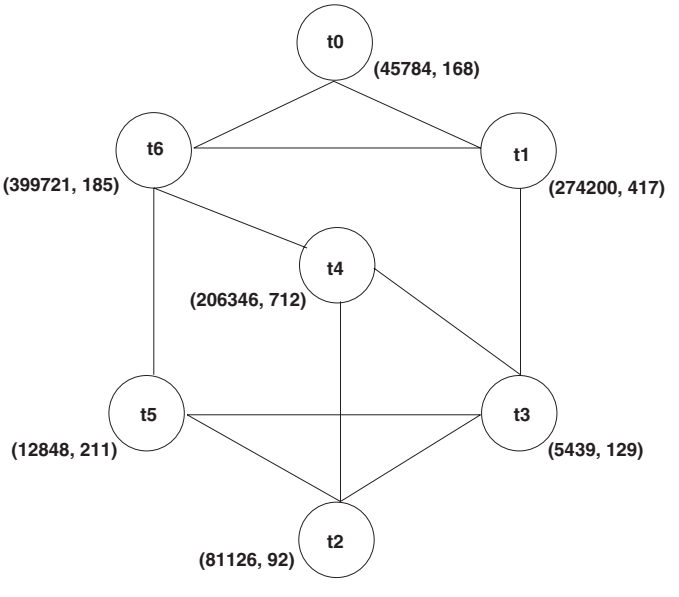

Fig. 10. Compatibility graph for the Larsson SOC.[15]

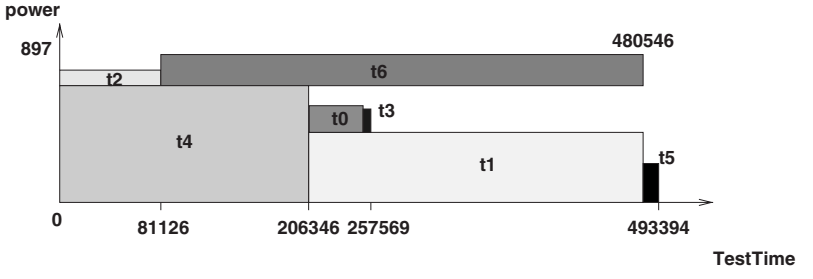

Fig. 11. Solution for the Larsson SOC.

implemented by Zhao and Upadhyaya[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[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[10] test scheduled this example using an MILP formulation in 7985 cycles *with a* power constraint of $P_{\max} = 950\,\text{mW}$. 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.

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

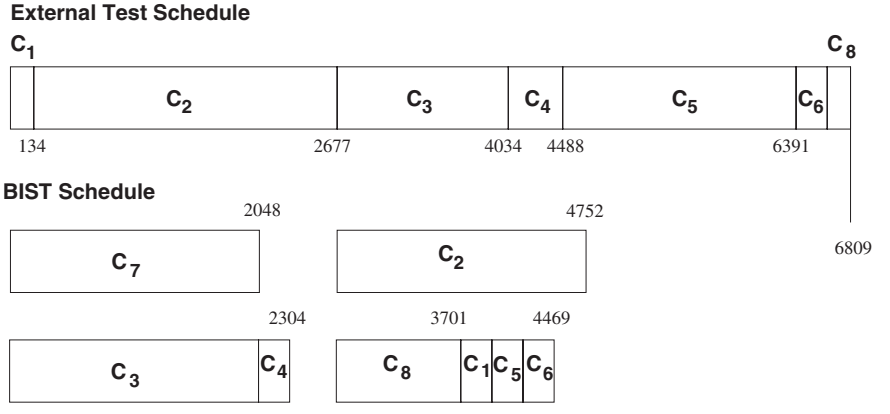

Fig. 12. Test schedule for d5018.

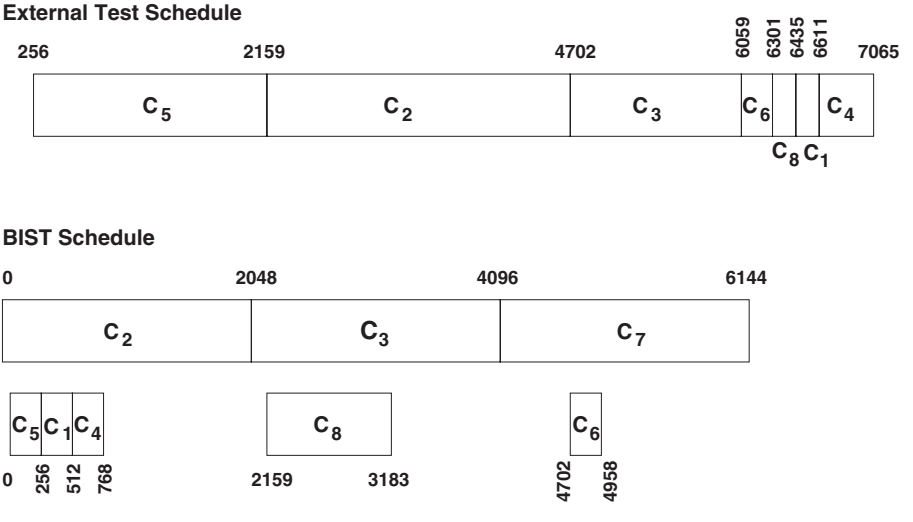

Fig. 13. Test schedule for d5018 under power constraints.

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*

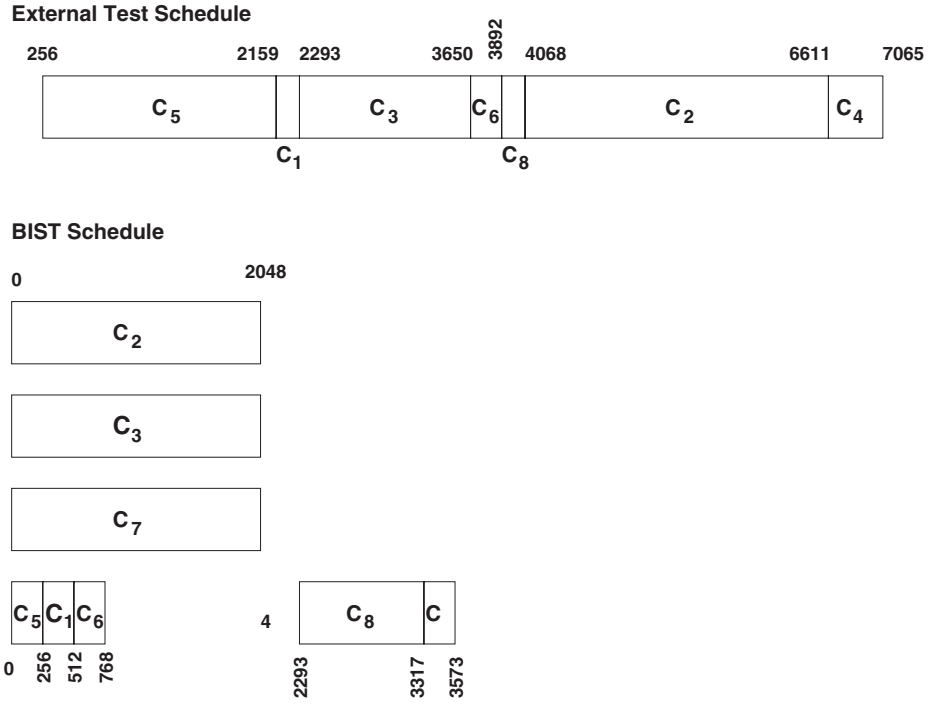

Fig. 14. Test schedule for d5018 under precedence constraints on BIST and external tests.

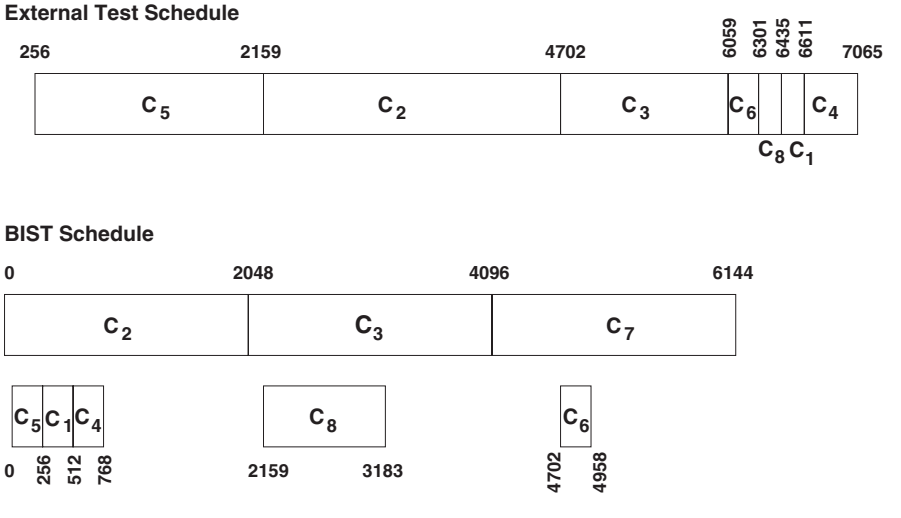

Fig. 15. Test schedule for d5018 using power and precedence constraints.

Table 5. Test data for the d5018 system.

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

Table 6. Test scheduling results for the d5018 SOC.

Scheduling constraints	Approach	Test Time	Difference to Optimum (%)
Resource	Optimal	6809	—
	Chakrabarty[2]	7851	15.30
	Proposed approach	6809	0
Power (950 mW)	Optimal	6809	—
	Chakrabarty[2]	7851	15.30
	Iyengar and Chakrabarty[10]	7985	17.27
	Flottes *et al.*[6]	6809	0
	Proposed approach	6809	0
Precedence on BIST and external test constraints	Optimal	7065	—
	Chakrabarty[2]	7065	0
	Proposed approach	7065	0
Power, precedence, and test constraints	Optimal	7065	—
	Chakrabarty[2]	8723	23.47
	Proposed approach	7065	0

10, but fail to do so for the Flottes SOC.[6] The same observation can be made in Table 6 where Chakrabarty[2] 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

1. M. Bushnell and V. Agrawal, *Essentials of Electronic Testing for Digital, Memory and Mixed-Signal VLSI Circuits* (Kluwer-Academic Publishers, 2000).
2. K. Chakrabarty, Test scheduling for core-based systems using mixed-integer linear programming, *IEEE Trans. Computer-Aided Des.* **19**(10) (2000) 1163–1174.
3. K. Chakrabarty, Test scheduling for core-based systems, in *Proc. Int. Conf. Computer-Aided Des.*, San Jose, CA, 1999, pp. 391–394.
4. R. Chou, K. Saluja and V. Agrawal, Scheduling tests for VLSI systems under power constraints, *IEEE Trans. VLSI* **5**(2) (1997) 175–185.
5. G. L. Craig, C. R. Kime and K. K. Saluja Test scheduling and control for VLSI built-in self-test, *IEEE Trans. Comput.* **37**(9) (1988) 1099–1109.
6. M.-L. Flottes, J. Pouget and B. Rouzeyre, Power-constrained test scheduling for SoCs under a no session, in *SoC Design Methodologies*, eds. M. Robert, B. Rouzeyre, C. Piguet and M.-L. Flottes (2002), pp. 401–412.
7. T. Gonzales and S. Sahni, Open shop scheduling to minimize finish time, *J. ACM* **23** (1976) 665–679.
8. H. Harmanani and H. Salamy, Power-constrained system-on-a-chip test scheduling using a genetic algorithm, *J. Circuits Syst. Comput.* **15**(3) (2006) 331–349.
9. Y. Huang, W.-T. Cheng, C.-C. Tsai, N. Mukherjee, O. Samman, Y. Zaidan and S. Reddy, Resource allocation and test scheduling for concurrent test of core-based SoC design, *Proc. ATS* (2001), pp. 265–270.
10. V. Iyengar and K. Chakrabarty, System-on-a-chip test scheduling using precedence relationships, preemptive, and power-constraints, *IEEE Trans. Comput.-Aided Des.* **21**(9) (2002) 1088–1094.
11. V. Iyengar, K. Chakrabarty and E. Marinissen, Test wrapper and test access mechanism co-optimization for system-on-a-chip, in *Proc. Int. Test Conf.* Baltimore (2001), pp. 1023–1032.
12. V. Iyengar, K. Chakrabarty and E. Marinissen, Efficient wrapper/tam co-optimization for large socs, in *Proc. Des. Automat. Test Eur. (DATE)*, Paris (2002), pp. 491–498.
13. E. Larsson, A. Larsson and Z. Peng, Linkoping University SOC Test Site. http://www.ida.liu.se/labs/eslab/soctest/klas/.
14. E. Larsson and Z. Peng, An integrated system-on-chip test framework, *Proc. Des. Automat. Test Eur.* (2001), pp. 138–144.
15. E. Larsson and Z. Peng, Test scheduling and scan-chain division under power constraint, *Proc. Asian Test Symp.* (2001), pp. 259–264.
16. E. Larsson and Z. Peng, A reconfigurable power-conscious core wrapper and its application to SOC test scheduling, *Proc. Int. Test Conf.* (2003), pp. 1135–1144.
17. E. Larsson, Z. Peng and G. Carlsson, The design and optimization of SOC test solutions, *Proc. ICCAD* (2001), pp. 523–530.
18. V. Muresan, X. Wang, V. Muresan and M. Vladutiu, A comparison of classical scheduling approaches in power-constrained block-test scheduling, *Proc. Int. Test Conf.* (2000), pp. 882–891.

19. J. Pouget, E. Larsson and Z. Peng, SOC test minimization under multiple constraints, in *Proc. Asian Test Symp.*, China (2003), pp. 312–317.
20. C. Ravikumar, G. Chandra and A. Verma, Simultaneous module selection and scheduling for power constrained testing of core based systems, *Proc. VLSI Design*, (2000).
21. M. Vecchi S. Kirkpatrick and C. Gelalt, Optimization by simulated annealing, *Science* **220**(4598) (1983) 671–680.
22. C. Su and C. Wu, A graph-based approach to power-constrained test scheduling, *J. Electron. Testing: Theory Appl.* **20** (2004) 45–60.
23. M. Sugihara, H. Date and H. Yasuura, A novel test methodology for core-based system LSIs and a testing time minimization problem, *Proc. Int. Test Conf.* (1998), pp. 465–472.
24. D. Zhao and S. Upadhyaya, Adaptive test scheduling in SoC's by dynamic partitioning, in *Proc. Int. Symp. Defect Fault Tolerance VLSI Syst.* (2002), pp. 334–342.
25. D. Zhao and S. Upadhyaya, A generic resource distribution and test scheduling scheme for embedded core-based SoCs, *IEEE Trans. Instrum. Meas.* **53**(2) (2004) 318–329.
26. D. Zhao and S. Upadhyaya, Dynamically partitioned test scheduling for SoCs under power constraints, *IEEE Trans. CAD* **24**(6) (2005) 956–965.
27. Y. Zorian, A distributed BIST control scheme for complex VLSI devices, *Proc. 11th IEEE VLSI Test Symp.* (1993), pp. 4–9.