Short Papers

System-on-a-Chip Test Scheduling With Precedence Relationships, Preemption, and Power Constraints

Vikram Iyengar and Krishnendu Chakrabarty

***Abstract*—Test scheduling is an important problem in system-on-a-chip (SOC) test automation. Efficient test schedules minimize the overall system test application time, avoid test resource conflicts, and limit power dissipation during test mode. In this paper, we present an integrated approach to several test scheduling problems. We first present a method to determine optimal schedules for reasonably sized SOCs with precedence relationships, i.e., schedules that preserve desirable orderings among tests. We also present an efficient heuristic algorithm to schedule tests for large SOCs with precedence constraints in polynomial time. We describe a novel algorithm that uses preemption of tests to obtain efficient schedules for SOCs. Experimental results for an academic SOC and an industrial SOC show that efficient test schedules can be obtained in reasonable CPU time.**

***Index Terms*—Core-based systems, embedded core testing, mixed-integer linear programming, open shop schedules, precedence constraints, preemptive test scheduling.**

I. Introduction

Predesigned and preverified intellectual property (IP) cores are being increasingly used in complex system-on-a-chip (SOC) designs. However, testing these systems is difficult, and manufacturing test is widely recognized as a major bottleneck in SOC design [15]. A major challenge confronting the system integrator is test scheduling, which determines the order in which the various cores are tested. A combination of built-in self-test (BIST) and external testing must often be used to achieve high fault coverage [2], [13]. An effective test scheduling approach must minimize testing time while addressing the following issues: 1) resource conflicts between cores arising from the use of shared TAMs and on-chip BIST engines; 2) precedence constraints among tests; and 3) power dissipation constraints.

Test scheduling for SOCs is especially challenging since even a simple SOC test scheduling problem is equivalent to the $\mathcal{NP}$-complete m-processor open shop scheduling problem [2]. (We formally define this problem in Section II.) Most recent test scheduling techniques for SOCs use heuristics that address only certain aspects of the problem. These include selecting the best test for a core from a set of potential tests supplied by the core vendor [13], approximate vertex cover of a resource-constrained test compatibility graph with a view to limit power consumption [5], and reordering of tests to detect defects earlier during manufacturing test [9]. The use of test protocols [11], tree-growing algorithms for power-constrained scheduling [12], and integrated TAM design and test scheduling [10] are other recently proposed techniques that address certain aspects of the test scheduling problem. All of the above approaches are based on heuristics and do not guarantee optimal solutions. Moreover, they concentrate only on a few aspects of the problem and do not provide an integrated approach to address all facets of test scheduling.

Optimal test scheduling was the focus of a recent paper, which presented scheduling techniques for cores that require a combination of external scan vectors and BIST [2]. However, in this work, each core could only be accessed by a single test bus and one BIST engine. Furthermore, all the external tests for the SOC were scheduled on a single test bus. Moreover, the problems of scheduling tests with precedence constraints, test preemption, and power minimization were not considered in [2]. *Precedence* constraints impose a partial order among the tests in a test suite. This can be motivated by several factors. For example, since BIST is likely to detect more defects than an external test targeted only at random-resistant faults, it may be desirable to apply BIST first to a core during manufacturing test. Similarly, it may be desirable to test and diagnose memories earlier so that they can be used later for system test. Since larger cores are more likely to have defects (due to their larger silicon area), it may also be more desirable to test them first. Furthermore, in practice tests are often reordered based on test results obtained from a small quality assurance sample. This information can be exploited to reorder tests such that potential low-yield modules in the final high-volume production batch are tested first. Embedding such precedence constraints in the test schedule can play an important role in increasing the overall efficiency of a test suite.

Preemptive test scheduling offers low testing time and significantly lower computational complexity than exact methods, at the expense of test application overhead. Preemptive tests can be halted for a period of time and then restarted, similar to the blocked multithreading approach used in multipipeline microprocessors. A major factor motivating the use of test preemption is that preemptive test schedules can be obtained in polynomial time, thus greatly reducing computation time, especially since the general (nonpreemptive) scheduling problem is $\mathcal{NP}$-complete [2].

SOCs in test mode can dissipate up to twice the amount of power they do in normal mode, since cores that do not normally operate in parallel may be tested concurrently to minimize testing time [12]. *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. A realistic integrated approach to test scheduling for SOCs, therefore, mandates the inclusion of precedence, test preemption, and power constraints in the test schedule.

In this paper, we present a new integrated approach to SOC test scheduling. The proposed approach obtains optimal test schedules with precedence constraints for reasonably sized SOCs. For precedence-based scheduling of large SOCs, we develop an efficient algorithm that has a run time of $O(n^2)$, where n is the number of tests in the test suite. The proposed integrated scheduling approach also includes an algorithm to obtain preemptive test schedules in $O(n^3)$ time, where n is the number of tests. Preempting external tests applied by the ATE requires that the test patterns stored in the ATE memory be reordered. Preempting a BIST test requires the BIST engine to save LFSR and MISR states, and preempting the test for sequential cores requires storage of internal flip–flop states. Therefore, it is desirable to keep preemptions to a minimum; in particular, savings in idle times need to be carefully weighed against the time needed to reinitialize the BIST engines and

Manuscript received April 11, 2001; revised December 22, 2001. This work was supported in part by the National Science Foundation under Grant CCR-9875324 and in part by an equipment grant from Intel Corporation. This paper was published in part in *Proc. IEEE VLSI Test Symp.*, pp. 368–374, April–May, 2001. This paper was recommended by Associate Editor R. Gupta.

V. Iyengar was with the Department of Electrical and Computer Engineering, Duke University, Durham, NC 27708 USA. He is now with IBM Microelectronics, Essex Junction, VT 05452 USA (e-mail: vikrani@us.ibm.com).

K. Chakrabarty is with the Department of Electrical and Computer Engineering, Duke University, Durham, NC 27708 USA (e-mail: krish@ee.duke.edu).

Publisher Item Identifier 10.1109/TCAD.2002.801102.

0278-0070/02$17.00 © 2002 IEEE

TABLE I
RANGES IN TEST DATA FOR THE CORES IN d5018 AND p93791

Example SOC	Number range				Scan chain lengths	
	External test patterns	BIST patterns	Functional I/Os	Scan chains	Min	Max
d5018	26–158	256–2048	28–221	2–40	16	29
p93791	11-916	42–6127	21–903	11–46	1	500

the sequential elements in a core. Our test preemption algorithm incorporates parameters that allow only a certain number of preemptions per test; excessive BIST and sequential circuit test preemptions are prevented. Finally, we present a new power-constrained scheduling technique that is a significant improvement over [2]. The proposed power constraints can be easily embedded in the scheduling framework in combination with the test precedence constraints, thus delivering an integrated approach to the SOC test scheduling problem.

As in [8] and [14], we assume a TAM architecture based on multiple test buses. Details on test bus design and optimization are presented in [4] and [8]. Each core may be tested by several external test buses and/or BIST engines. However, the cores on each test bus or BIST engine are tested sequentially, and a given core can be accessed by only one test bus and/or BIST engine at a time. Furthermore, the external test for a given core cannot be applied at the same time as its BIST test. The test buses and BIST engines operate independently of each other. Thus each test bus and BIST engine is able to apply a test independently of the other test buses and BIST engines.

A. Example SOCs

We now introduce two example SOCs, d5018 and p93791, that will be used as running examples throughout this paper. SOC d5018 is an example SOC that consists of eight ISCAS benchmark cores. SOC p93791 is a large industrial SOC from Philips. This SOC consists of 32 embedded cores (14 logic cores and 18 memory cores embedded within logic cores). We assume that the memory cores are tested using BIST. The number, e.g., "5018" in each SOC name is a measure of its test complexity. This number is calculated using the formula presented in [8]. The letters "d" and "p" refer to Duke University and Philips, respectively. Table I presents a summary of the test data for the two SOCs.

II. PRECEDENCE-BASED SCHEDULING

Precedence constraints impose a partial order among the tests in a test suite. Here, we first present the formal definition for the m-processor open shop scheduling problem, as given in [6], and then formulate the precedence-based SOC test scheduling problem that we have examined.

- $\mathcal{P}_{\textbf{open shop}}$ [6]: Given $m \in Z^+$ processors, a set I of jobs, each job $i \in I$ consisting of m tasks $T_1[i], T_2[i], \ldots, T_m[i]$ (with $T_j[i]$ to be executed by processor j), a length $l[T] \in Z_0^+$ for each task T, and an overall deadline $D \in Z^+$, is there an open shop schedule for J that meets the deadline, i.e., a collection of one-processor schedules σ_j: $I \to Z_0^+$, $1 \le j \le m$, such that $\sigma_j(i) > \sigma_j(k)$ implies $\sigma_j(i) \ge \sigma_j(k) + l(T_j[k])$, and for each $i \in I$ the intervals $[\sigma_j(i), \sigma_j(i) + l(T_j[j]))$ are all disjoint, and such that $\sigma_j(i) + l(T_j[i]) \le D$, for $1 \le j \le m$, $1 \le i \le |I|$?

It was shown in [6] that $\mathcal{P}_{\text{open shop}}$ is $\mathcal{NP}$-complete for $m \ge 3$. Based on the formal definition of open shop scheduling [6], we next define a simple case of the SOC test scheduling problem as follows.

- $\mathcal{P}_{\text{simple}}$: Given a set of N_C cores, a set of test resources consisting of multiple test buses and BIST engines, and a suite of tests such that each core requires one or more tests, is there a test schedule such that: 1) no two tests for the same core are applied concurrently; 2) there are no test resource conflicts; and 3) the schedule meets an overall deadline D?

Let the set of cores in $\mathcal{P}_{\text{simple}}$ be denoted by I, such that $|I| = N_C$. Further, let the number of test resources be m, such that each core $i \in I$ has m tests $T_1[i], T_2[i], \ldots, T_m[i]$ (with $T_j[i]$ to be generated by test resource j). If each test T has length $l[T] \in Z_0^+$, then the decision problem $\mathcal{P}_{\text{simple}}$ reduces to the m-processor open shop scheduling problem and is $\mathcal{NP}$-complete for $m \ge 3$ [6]. An algorithm to solve $\mathcal{P}_{\textbf{open shop}}$ in polynomial time for $m = 2$ was presented in [7]. Based on this work, an algorithm to schedule tests for SOCs having a single test bus and a single BIST engine was developed in [2]. The complexity of this algorithm was shown to be $O(r)$, where r is the number of cores. However, most SOCs contain multiple test buses and BIST engines (including cores with dedicated BIST engines), and therefore the practical case of the scheduling problem remains intractable.

We now examine the problem of scheduling tests with precedence constraints. A general case of the precedence-based scheduling problem, in which any BIST or external test may be ordered before any other, can be stated as follows.

- $\mathcal{P}_{\text{prec_general}}$: Given a set of N_C cores, a set of test resources consisting of multiple test buses and BIST engines, and a suite of tests such that each core requires one or more tests, is there a test schedule such that: 1) no two tests for the same core are applied concurrently; 2) there are no test resource conflicts; 3) the schedule incorporates precedence constraints ($T_j[i] < T_l[k]$), such that test $T_j[i]$ is applied before test $T_l[k]$; and 4) the schedule meets an overall deadline D?

Special cases of $\mathcal{P}_{\text{prec_general}}$ include the cases where: 1) BIST for cores is applied first; 2) certain larger cores are tested first; and 3) cores purchased from external vendors are tested before cores designed in-house. It can be easily shown that $\mathcal{P}_{\text{prec_general}}$ with m test resources is equivalent to the m-processor open shop decision problem described in [7], by restriction to special case. Thus $\mathcal{P}_{\text{prec_general}}$ is also $\mathcal{NP}$-complete for $m \ge 3$ [7]. We next develop a mixed-integer linear programming (MILP) model to solve $\mathcal{P}_{\text{prec_general}}$ exactly for SOCs of moderate size.

We now introduce the notation that we use to represent SOC tests and their start times in the remainder of this paper. Each test is represented by a symbol T_p, where $p > 0$ is a unique integer. We use t_p to denote the start time of test T_p. Now, let t_{2i-1} and t_{2i} denote the start times of the BIST test T_{2i-1} and external test T_{2i}, respectively, for core i, $1 \le i \le N_C$. Similarly, let $l_{2i-1} \ge 0$ and $l_{2i} \ge 0$ denote the lengths of the BIST and external tests for core i, respectively. Note that this (t_{2i-1}, t_{2i}) notation assumes that each core has two or less tests. In case a core has more than two tests, this notation will need to be extended. For example, if each core has three or fewer tests, t_{3i-2}, t_{3i-1}, and t_{3i} denote the start times of the three tests for Core i.

Two tests T_i and T_j overlap if: $t_i < t_j + l_j$ and $t_i + l_i > t_j$. On the other hand, two tests T_i and T_j do not overlap if and only if either: 1) $t_i - t_j - l_j \ge 0$ or 2) $t_j - t_i - l_i \ge 0$. Two tests have a conflict if they cannot be applied at the same time. This can occur if: 1) they are applied by the same test resource or 2) they are tests for the same core. We introduce a binary parameter x_{ij}, $1 \le i, j \le 2N_C$, to represent conflicts between tests T_i and T_j as follows:

$$x_{ij} = \begin{cases} 1, & \text{if tests } T_i \text{ and } T_j \text{ are conflicting} \\ 0, & \text{otherwise.} \end{cases}$$

We can now formulate a mathematical programming model to solve problem $\mathcal{P}_{\text{prec_general}}$ as follows.

Objective: Minimize $\mathcal{T} = \max_{i \in \{1, 2, \ldots, 2N_C\}} \{t_i + l_i\}$ subject to:

1) $x_{ij}(t_i - t_j - l_j) \ge 0$ **or** $x_{ij}(t_j - t_i - l_i) \ge 0$, where $1 \le i, j \le 2N_C$;
2) $t_q \ge t_p + l_p$, for each precedence constraint ($T_p < T_q$).

The above nonlinear cost function $\mathcal{T}$ for $\mathcal{P}_{\text{prec_general}}$ can be easily linearized by minimizing $\mathcal{T}$ and adding the constraints $\mathcal{T} \geq t_i + l_i$ for $1 \leq i \leq 2N_C$. The **or** construct in the first constraint can be easily linearized as shown in [2]. We introduce 0–1 indicator variables δ_{ij1} and δ_{ij2}, $1 \leq i, j \leq 2N_C$, to the set of constraints. The optimization model is now restated as follows.

Objective: Minimize $\mathcal{T}$ subject to:

1) $\mathcal{T} \geq t_i + l_i$, $1 \leq i \leq 2N_C$;
2) $x_{ij}\delta_{ij1}(t_i - t_j - l_j) + x_{ij}\delta_{ij2}(t_j - t_i - l_i) \geq 0$;
3) $\delta_{ij1} + \delta_{ij2} = 1$;
4) $t_q \geq t_p + l_p$.

The second constraint above can be linearized by replacing $t_i\delta_{ij1}$ by a new real variable s_{ij1} and $t_j\delta_{ij1}$ by s_{ij2}. Similarly, we replace $t_j\delta_{ij2}$ by s_{ij3} and $t_i\delta_{ij2}$ by s_{ij4}. Each substitution requires the addition of three constraints. For example, to replace $t_i\delta_{ij1}$ by s_{ij1}, we add the following constraints:

1) $s_{ij1} - M\delta_{ij1} \leq 0$;
2) $-t_i + s_{ij1} = 0$;
3) $t_i - s_{ij1} + M\delta_{ij1} \leq M$

where $M = \sum_{i=1}^{2N_C} l_i$ is an upper bound on the value of t_i. The resulting MILP model is as follows.

Objective: Minimize $\mathcal{T}$ subject to:

1) $\mathcal{T} \geq t_i + l + i$, $1 \leq i \leq 2N_C$;
2) $x_{ij}(s_{ij1} - s_{ij2} - l_j\delta_{ij1}) + x_{ij}(s_{ij3} - s_{ij4} - l_i\delta_{ij2}) \geq 0$;
3) $\delta_{ij1} + \delta_{ij2} = 1$;
4) $s_{ij1} - M\delta_{ij1} \leq 0$;
5) $-t_i + s_{ij1} = 0$;
6) $t_i - s_{ij1} + M\delta_{ij1} \leq M$;
7) $t_q \geq t_p + l_p$, where p, q represent the tests applied to the various cores.

We first solved the $\mathcal{P}_{\text{simple}}$ scheduling problem for SOC d5018. Fig. 1(a) illustrates an optimal schedule for d5018. For simplicity, we assume a test architecture containing one test bus and four BIST engines. The correspondence of core tests to test resources was provided in the input file to the $lpsolve$ linear programming tool [1], which took 3 s to obtain this optimal schedule. Note that the test lengths in Fig. 1 are not drawn to scale. Next we used the MILP model to solve $\mathcal{P}_{\text{prec_general}}$ for d5018. For the purposes of illustration, we incorporated precedence constraints into the scheduling model, such that the BIST test for each core is applied before its external test. This is often the practice when memory cores must be tested and diagnosed using BIST before system test is run. The precedence constraints added to the $\mathcal{P}_{\text{prec_general}}$ model were as follows: $\{(T_1 < T_2), (T_3 < T_4), (T_5 < T_6), (T_7 < T_8), (T_9 < T_{10}), (T_{11} < T_{12}), (T_{15} < T_{16})\}$. Here the external test for each Core i is represented by T_{2i-1} and its BIST test is represented by T_{2i}. A constraint $(T_i < T_j)$ denotes that test T_i is applied before T_j.

The CPU time to solve $\mathcal{P}_{\text{prec_general}}$ with the above precedence constraints was only 90 s. Fig. 1(b) illustrates the optimal schedule for d5018 obtained from our model, with BIST for each core scheduled before its external test. Compared to $\mathcal{P}_{\text{simple}}$, there is an increase in testing time of 256 clock cycles due to the time required to apply the BIST test of Core 4. This happens because the external tests may only begin after at least one BIST test completes. Thus the external test for Core 1 can no longer begin at time zero, as was the case in Fig. 1(a).

As stated earlier, the $\mathcal{P}_{\text{prec_general}}$ scheduling problem for three or more processors (test resources) is $\mathcal{NP}$-complete. Therefore, while MILP may provide exact solutions for reasonably sized SOCs, the computation time grows exponentially with the number of cores and test resources. We next present a heuristic algorithm *Precede* to solve $\mathcal{P}_{\text{prec_general}}$ in polynomial time.

A pseudocode description of the *Precede* algorithm is provided in Fig. 2. The set of test resources including external test buses and BIST engines is $\mathbf{TR}$, such that $m = |\mathbf{TR}|$. $\mathbf{T}$ is the set of tests to be scheduled. The test for Core i on test resource j is represented by T_{ij}. The algorithm begins by sorting the tests on each test resource in increasing order of length. Next, the test resources are considered in a cyclical manner, beginning with the first test resource. The algorithm searches for the first test that can be scheduled, such that no precedence constraints are violated and there are no test resource conflicts. If such a test can be found, it is scheduled, else the next test resource is considered. The loop repeats until all the tests have been scheduled. The complexity of the algorithm is $O(n^2)$, since to schedule each of the n tests, $O(n)$ precedence checks with the remaining tests are required.

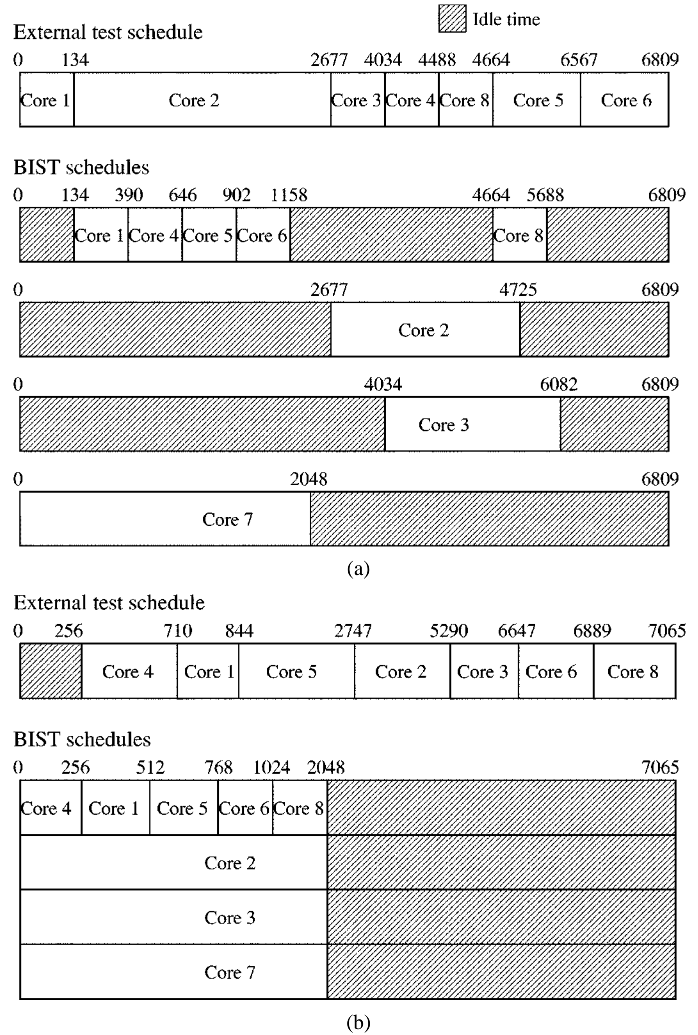


Fig. 1. Optimal (a) $\mathcal{P}_{\text{simple}}$ and (b) $\mathcal{P}_{\text{prec_general}}$ test schedules for SOC d5018.

We used *Precede* to perform test scheduling with precedence constraints for SOC p93791. Again, for the purpose of illustration, we incorporated precedence constraints such that the BIST test for the embedded memories lying within each logic core are tested before the external test for the logic core is applied. The resulting test schedule (obtained in less than a second of CPU time) is presented in Fig. 3. The total testing time is 520688 clock cycles. To obtain the schedule presented in Fig. 3, we assumed a test architecture consisting of five external test buses of width 12 bits each and two BIST engines. From Fig. 3, it is evident that test bus 3 (for Cores 6, 12, and 19) is the bottleneck test bus. In order to measure the increase in testing time due to precedence constraints, we next redesigned the test architecture to consist of six external test buses and two BIST engines. We reassigned the

Procedure *Precede*(**TR**, **T**)

1. **For** $j = 1$ to m
2. Sort all the tests for test resource j in increasing order of test length;
3. Set $j = 1$;
4. **While** $\mathbf{T} \neq \emptyset$
5. **For** each Test T_{ij} in increasing order of length
6. **If** (there is a precedence constraint ($T_{xy} < T_{ij}$) AND Test T_{xy} has completed) OR (there is no precedence constraint ($T_{xy} < T_{ij}$))
7. **If** some other Test T_{ik} is not currently scheduled
8. Schedule T_{ij}; $\mathbf{T} = \mathbf{T} - \{T_{ij}\}$;
9. **If** $j = m$, set $j = 1$;
10. **Else** set $j = j + 1$;

Fig. 2. The *Precede* algorithm.

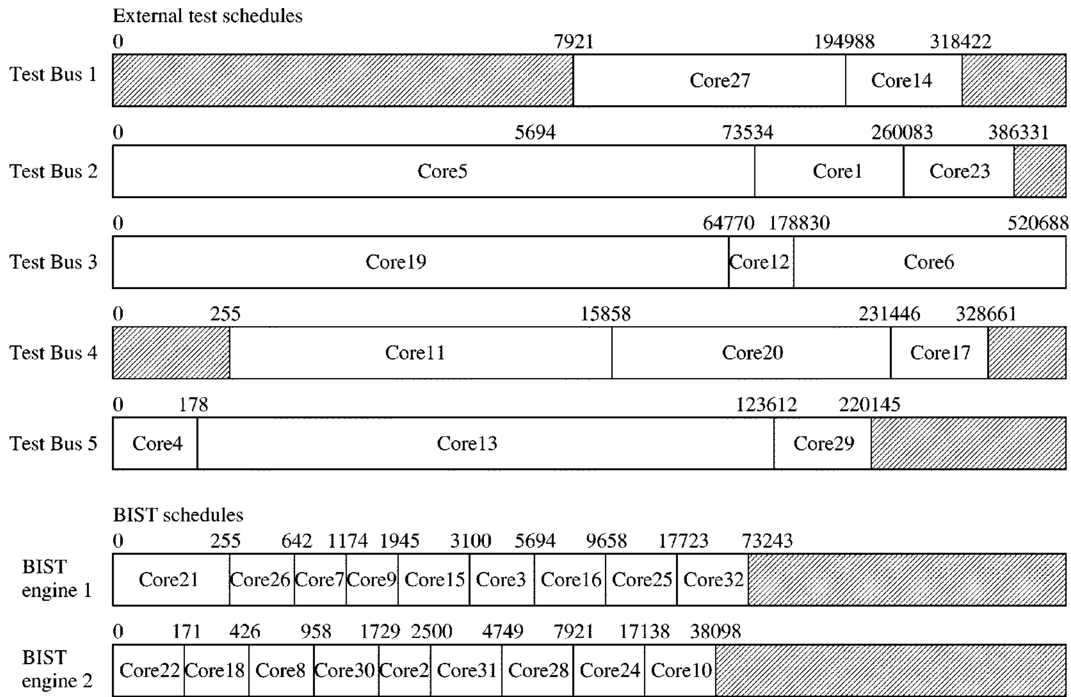


Fig. 3. Test schedule for SOC p93791 obtained using *Precede*.

cores to the test buses, such that no one test bus is the bottleneck. Next, we varied the total test bus width of the external test buses and performed test scheduling, both with and without precedence constraints. The testing time of Core i on a test bus of width w_j was calculated according to the formula presented in [8]. The values of total testing time obtained for several values of total external test bus width W are presented in Table II. The increase in testing time due to precedence constraints is on average 20%. The difference between testing times with and without precedence constraints increases as W is increased. With wider test buses, the tests are shorter and more closely packed; thus any constraint on ordering has a greater effect on testing time.

TABLE II
COMPARISON OF TESTING TIMES WITH AND WITHOUT PRECEDENCE CONSTRAINTS FOR SOC p93791

Total test bus width	Testing time $\mathcal{T}$ (cycles)		Percent increase in $\mathcal{T}$
	Without precedence	With precedence	
12	2681532	2995271	11.7
18	1880460	2126800	13.1
24	1375695	1616442	17.4
30	1128943	1266674	12.2
36	967017	1088861	12.6
42	802964	989252	23.2
48	796421	1020215	28.1
54	654703	827545	26.4
60	590976	752312	27.3

III. PREEMPTIVE SCHEDULING

While the m-processor nonpreemptive open shop scheduling problem is known to be $\mathcal{NP}$-complete for $m \geq 3$, optimal preemptive schedules for the open shop problem can be obtained in polynomial time for $m \geq 3$ [7]. This can significantly reduce the computation time for scheduling over exact methods such as MILP. Furthermore, preemptive schedules can obtain lower testing times than heristic methods for nonpreemptive scheduling, since tests are preempted to avoid resource conflicts that can cause idle times on test buses. In this section, we present an algorithm to obtain such preemptive test schedules in $O(n^3)$ time, where n is the number of tests.

Making external tests preemptive does not significantly increase the ATE's control complexity since the test suite to be stored is the same as before, except that the subsequences of test patterns are reordered in

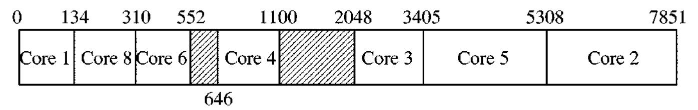
Procedure *Preempt*()

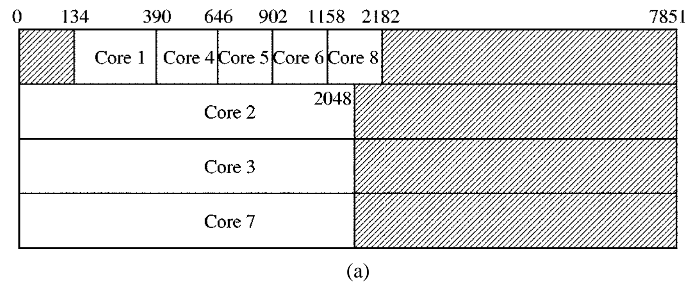
1. **For** each Core i
2. **For** each test resource j
   /* l_{ij} = length of test applied by test resource j to core i */
   /* $maxp_{ij}$ = maximum number of preemptions allowed */
3. Matrix element $m_{ij} = (l_{ij}, maxp_{ij})$;
4. **If** Test T_{ij} is a BIST test or a sequential circuit test
5. Set $maxp_{ij} = 0$;
6. Set $P = \Phi$; /* Initialize set P */
7. **While** matrix M contains a non-zero element
8. **For** each Core i
9. **For** each test resource j
10. **If** P has no elements from Core i or test resource j
11. Set $P = P \bigcup m_{ij}$; /* Add m_{ij} to P */
12. Let $m_{min} = \min_{ij}\{l_{ij}\} : l_{ij} \in P$;
13. Schedule tests represented by elements in P for time m_{min};
14. **For** each element $m_{ij} \in P$
15. Subtract $l_{ij} = l_{ij} - m_{min}$;
16. **If** $maxp_{ij} > 0$
17. Subtract $maxp_{ij} = maxp_{ij} - 1$;
18. **If** test T_{ij} is a scan test
    /* s_{ij} = length of longest scan chain in core */
19. Add $l_{ij} = l_{ij} + s_{ij}$;
20. Remove element m_{ij} from set P;

Fig. 4. The *Preempt* algorithm.

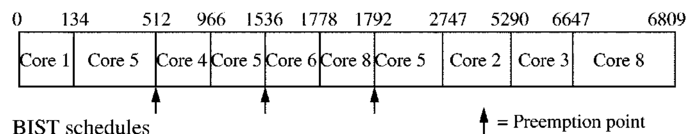
	Test resource j				
Core i	1	2	3	4	5
1	**134**	256			
2	2543		**2048**		
3	1357			**2048**	
4	454	**256**			
5	1903	256			
6	242	256			
7					**2048**
8	176	1024			

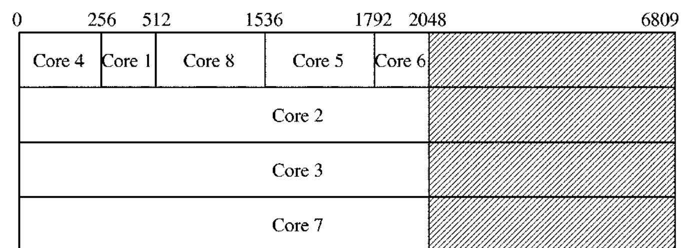
Fig. 5. Matrix M for preemptive scheduling of d5018.





(a)





(b)

Fig. 6. Test schedule for d5018 using (a) *Shortest-task-first* and (b) *Preempt*.

ATE memory. In addition, the short initializing sequence applied to the scheduling controller to switch a core between test and isolation modes will need to be reapplied. In a scan-based test, however, preemption after a scan operation will lead to a scan-out that cannot be overlapped with the next scan-in. Hence, testing time will increase by the time of one scan operation. Furthermore, to preempt a BIST test, the current LFSR state and intermediate MISR signature must be saved. Moreover, the states of internal flip–flops in sequential cores will need to be saved, if sequential core tests are preempted. Therefore, it is desirable to limit the number of BIST and sequential core test preemptions.

We present a polynomial-time preemptive scheduling method for SOCs, based in part on a preemptive open shop algorithm described in [7]. In this method, tests may be preempted by the algorithm in the search for the optimal test time. The algorithm accepts user-defined values for parameters that are to control the maximum number of preemptions allowed for each test. These parameters are also used to prevent preemption of BIST and sequential core tests during scheduling. Thus tests are preempted only when they result in a decrease in testing time at the cost of a small increase in hardware overhead.

In the proposed *Preempt* algorithm, we use a matrix representation of the SOC test schedule. The pseudocode for this algorithm is presented in Fig. 4. Each element m_{ij} in the matrix M is a 2-tuple containing: 1) the test length l_{ij} and 2) a limit $\max p_{ij}$ on the maximum number of preemptions allowed for each test T_{ij}. The value of l_{ij} initially equals the time taken to test Core i by test resource j. A set P of elements is selected to determine the first set of tests to be scheduled at time zero. Elements in the same row of M represent tests that have resource conflicts with each other, as do elements in the same column of M; therefore, two tests represented by elements in the same row or column of M are not scheduled together. The elements in P are decreased by the value of the smallest element in P. The next set of elements is selected and the process continues until all elements reduce to zero, i.e., all the tests are scheduled. A change in members of P between iterations represents a preemption. If preemptions are allowed, the $\max p_{ij}$ number associated with the preempted test is decremented by 1 each time a preemption occurs. Once the variable $\max p_{ij}$ goes to zero, i.e., the maximum number of preemptions have been reached, test T_{ij} must remain in set P in each consecutive iteration until it completes. Furthermore, each time a scan-based test is preempted, the test length l_{ij} is incremented by the number of cycles required for one scan operation s_{ij}.

The complexity of *Preempt* is calculated as follows. The **While** loop in Line 7 of *Preempt* executes n times (once for each time a test completes). For each such execution, the number of matrix elements evaluated by the nested **For** loops in Lines 8 and 9 is $O(n^2)$ in the worst case. Hence, the complexity of *Preempt* is $O(n^3)$. Further, the worst case number of preemptions per test resource is $O(n)$; this is because tests are preempted in Line 14, which executes a maximum of $O(n)$ times.

Fig. 5 illustrates the initial matrix M constructed by the *Preempt* algorithm to determine a preemptive schedule for d5018. The elements in bold represent the first set of tests scheduled at time zero. In Fig. 6,

TABLE III
COMPARISON OF TESTING TIMES WITHOUT AND WITH TEST PREEMPTION FOR SOC p93791

Total test bus width W	Testing time $\mathcal{T}$ (cycles)		Percent decrease in $\mathcal{T}$
	Without preemption	With preemption	
12	2681138	2669609	4.30
18	1881484	1712155	9.08
24	1373674	1289880	6.61
30	1123601	1026971	8.66
36	959448	853944	10.86
42	803988	733157	8.81
48	690727	641340	7.15
54	652358	570226	12.59
60	591852	513254	13.28.

we compare the testing time for d5018 obtained using the nonpreemptive scheduling algorithm *Shortest-task-first* [2] with the testing time obtained using *Preempt*. We compare our preemptive scheduling algorithm with *Shortest-task-first* because *Shortest-task-first* also has a complexity of $O(n^3)$. While the testing time obtained using *Shortest-task-first* for d5018 is 7851 cycles, the testing time using *Preempt* is 6857 clock cycles with three preemptions. This represents a savings of 12.7% in testing time over nonpreemptive scheduling.

Next, we conducted experiments on preemptive test scheduling for the industrial SOC p93791. For the purpose of illustration, we assumed that Core 6, Core 20, and Core 27 (the cores with the longest test sets) are sequential cores; therefore, their tests were not preempted. Further, no tests for memory cores were preempted. We obtained preemptive schedules for p93791 for several values of total external test bus width, as was done in Section II. The results on testing time using *Preempt* are compared with the testing times obtained using *Shortest-task-first* in Table III. The *Preempt* algorithm outperforms the *Shortest-task-first* algorithm for all values of W. Furthermore, savings in testing time using preemption increase from 4.3% for $W = 12$ to 13.28% for $W = 60$, thereby demonstrating the scalability of the proposed technique.

Both precedence-based and preemptive scheduling seek to minimize the overall testing time for the SOC. Power dissipation is not considered in these scheduling methods. However, power constraints are important because they limit test concurrency to ensure that the power rating of the SOC is not exceeded. In Section IV, we present a new method to model the concurrency between tests in a schedule and show how our model can be used to obtain test schedules that do not exceed the maximum power rating of the SOC.

IV. POWER-CONSTRAINED SCHEDULING

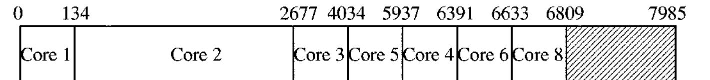
We first present a new method to model concurrency between tests for the cores. The constraints on test concurrency developed here can be easily embedded in the MILP model formulated in Section II. The enhanced MILP model can then be used to obtain test schedules that limit the amount of power dissipated during test for the $\mathcal{P}_{\text{simple}}$ and $\mathcal{P}_{\text{precedence}}$ scheduling problems.

We model test concurrency for power-constrained test scheduling as follows. Let P_i denote the power dissipated when test T_i alone is applied to the SOC. We define P_i to be the peak power dissipated over all test patterns in T_i, since this is the most realistic measure of power dissipation for the purpose of power-constrained test scheduling [5]. Let overlap parameter o_{ij} $(i < j)$ be defined as follows:

$$o_{ij} = \begin{cases} 1, & \text{if tests } T_i \text{ and } T_j \text{ overlap} \\ 0, & \text{otherwise.} \end{cases}$$

TABLE IV
POWER DISSIPATION IN TEST MODE FOR THE BIST TESTS

Core	Index i	Power P_i (mW)
c880	1	54
c2670	2	159
c7552	3	453
s953	4	57
s5378	5	324
s1196	6	72
s13207	7	792
s1238	8	75



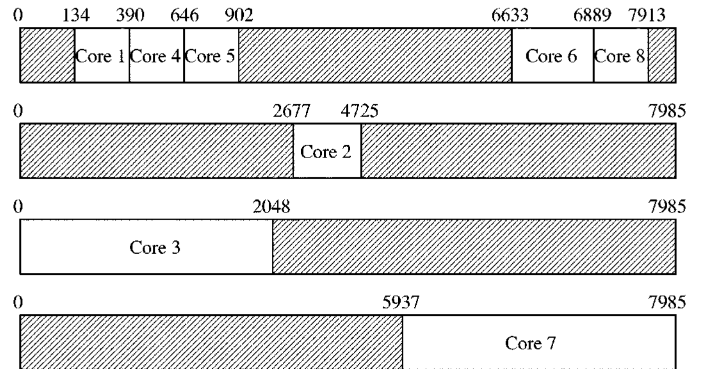


Fig. 7. Power-constrained test schedule for d5018.

Let t_i and t_j be the start times of tests T_i and T_j, respectively. The condition for overlap was stated in Section II as follows: Two tests T_i and T_j overlap $(o_{ij} = 1)$, if $t_i + l_i > t_j$ and $t_j + l_j > t_i$. To embed power constraints in our scheduling algorithm, we add the following two constraints to the MILP model described in Section II, in the form of the following constraints:

1) $o_{ij}(t_i + l_i - t_j) > 0$;
2) $o_{ij}(t_j + l_j - t_i) > 0$.

These two constraints imply that if parameter o_{ij} equals zero, i.e., tests T_i and T_j must not be allowed to overlap, then either: 1) T_i is begun after T_j completes or 2) T_j is begun after T_i completes.

Let the maximum power dissipation allowed during testing be $P_{\max}$. In practice, this value will be lower than the maximum power rating of the SOC by a safety margin. For two tests T_i and T_j, the constraint on test concurrency to limit power dissipation can be expressed as: $o_{ij}(P_i + P_j) \leq P_{\max}$, where T_i and T_j are scheduled on different test resources. This can be extended to any number of tests by noting that a set of tests scheduled on different test resources are concurrent, if they are pairwise overlapping. Therefore, to ensure that four different tests executing on different test resources do not overlap, we will require six constraints in the worst case, i.e., if all six combinations exceed the power consumption limit. However, fewer constraints will be required if only a few combinations of tests exceed $P_{\max}$. For the case of SOC d5018 having additional test resources, we added (linearized) constraints for only those groups of tests, for which the sum of the power dissipated exceeded $P_{\max}$. A hypothetical power dissipation amount P_i for each test T_i for SOC d5018 was calculated based on

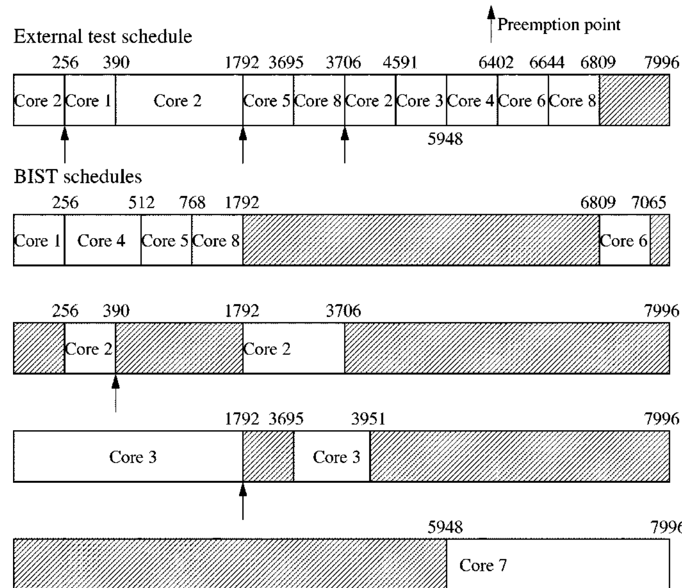


Fig. 8. Preemptive test schedule for d5018 incorporating precedence and power constraints.

the number of gates in the core to which T_i is applied. (More sophisticated power models can be easily incorporated.) These power dissipation values are shown in Table IV. $P_{\max}$ was chosen to be 950 mW.

The power-constrained test schedule obtained for d5018 is shown in Fig. 7. The CPU time taken by $lpsolve$ to obtain this schedule was 15 s. Note that this schedule is not provably optimal under power constraints; $lpsolve$ exited prematurely due to numerical instability after 15 s. Even though Cores 2 and 7 do not share a BIST engine, their BIST tests cannot be applied concurrently because of the 950-mW limit imposed on power dissipation. Similarly, the tests for Cores 3 and 7, and Cores 5 and 7 cannot be applied in parallel. The testing time for d5018, under power constraints, is 7985 clock cycles, which represents an increase of 16.9% over the optimal $\mathcal{P}_{\text{simple}}$ schedule. The 950-mW power constraint, when added to the $\mathcal{P}_{\text{precedence}}$ test scheduling model resulted in a testing time of 8723 clock cycles, an increase of 23.5% compared to the $\mathcal{P}_{\text{precedence}}$ case without power constraints.

Power constraints can also be added to the *Precede* and *Preempt* algorithms. For *Precede*, all the tests are first sorted in increasing order of test length (line 2 in Fig. 2). They are then scheduled on their respective resources in sorted order as soon as there is an available time at which the power dissipated by the currently scheduled tests does not exceed $P_{\max}$. To add power constraints to *Preempt*, a check is added to line 9 in Fig. 4 to ensure that the power dissipated by the tests in set P does not exceed $P_{\max}$.

We incorporated precedence and power constraints in the *Preempt* algorithm to obtain a preemptive test schedule for d5018 with a maximum power dissipation of $P_{\max} = 950$ mW and the following precedence constraints among individual tests: ($T_1 < T_2$), ($T_4 < T_8$), ($T_5 < T_6$), ($T_9 < T_{10}$), ($T_{16} < T_{11}$). The schedule is shown in Fig. 8. The total testing time is 7996 clock cycles. The testing time for the non-preemptive power-constrained schedule without precedence constraints obtained earlier is 7985 clock cycles—see Fig. 7. Thus, the potential savings in testing time using preemption have been offset by the increase in testing time due to the added precedence constraints in this schedule.

Acknowledgment

The authors would like thank E. J. Marinissen of Philips Research Laboratories for valuable comments on a draft version of this manuscript and for providing test data for the Philips SOC. The authors are grateful to Prof. B. Rouzeyre and J. Pouget of the Laboratoire d'Informatique de Robotique et de Microelectronique de Montpellier for pointing out an error in an earlier version of this manuscript. This work was conducted while V. Iyengar was at Duke University.

References

[1] M. Berkelaar. $lpsolve$ 3.0. Eindhoven Univ. Tech., Eindhoven, The Netherlands. [Online]. Available: ftp://ftp.ics.ele.tue.nl/pub/lp_solve.

[2] K. Chakrabarty, "Test scheduling for core-based systems using mixed-integer linear programming," *IEEE Trans. Computer-Aided Design*, vol. 19, pp. 1163–1174, Oct. 2000.

[3] ——, "Design of system-on-a-chip test access architectures under place-and-route and power constraints," in *Proc. Design Automation Conf.*, 2000, pp. 432–437.

[4] ——, "Optimal test access architectures for system-on-a-chip," *ACM Trans. Design Automation Electron. Syst.*, vol. 6, pp. 26–49, Jan. 2001.

[5] R. M. Chou, K. K. Saluja, and V. D. Agrawal, "Scheduling tests for VLSI systems under power constraints," *IEEE Trans. VLSI Syst.*, vol. 5, pp. 175–185, June 1997.

[6] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of $\mathcal{NP}$-Completeness*. San Francisco, CA: W. H. Freeman, 1979.

[7] T. Gonzalez and S. Sahni, "Open shop scheduling to minimize finish time," *J. Assoc. Computing Machinery*, vol. 23, no. 4, pp. 665–679, Oct. 1976.

[8] V. Iyengar, K. Chakrabarty, and E. J. Marinissen, "Test wrapper and test access mechanism co-optimization for system-on-chip," *J. Electron. Testing: Theory Applicat.*, vol. 18, pp. 213–230, Apr. 2002.

[9] W. Jiang and B. Vinnakota, "Defect-oriented test scheduling," in *Proc. VLSI Test Symp.*, 1999, pp. 433–438.

[10] E. Larsson and Z. Peng, "An integrated system-on-chip test framework," in *Proc. Design Automation and Test in Europe (DATE) Conf.*, 2001, pp. 138–144.

[11] E. J. Marinissen and M. Lousberg, "The role of test protocols in testing embedded-core-based system ICs," in *Proc. Eur. Test Workshop*, 1999, pp. 70–75.

[12] V. Muresan *et al.*, "A comparison of classical scheduling approaches in power-constrained block-test scheduling," in *Proc. Int. Test Conf.*, 2000, pp. 882–891.

[13] M. Sugihara, H. Date, and H. Yasuura, "A novel test methodology for core-based system LSIs and a testing time minimization problem," in *Proc. Int. Test Conf.*, 1998, pp. 465–472.

[14] P. Varma and S. Bhatia, "A structured test re-use methodology for core-based system chips," in *Proc. Int. Test Conf.*, 1998, pp. 294–302.

[15] Y. Zorian, E. J. Marinissen, and S. Dey, "Testing embedded-core-based system chips," *IEEE Comput.*, vol. 32, pp. 52–60, June 1999.