Test Scheduling for Core-Based Systems Using Mixed-Integer Linear Programming

Krishnendu Chakrabarty, *Member, IEEE*

***Abstract*—We present optimal solutions to the test scheduling problem for core-based systems. Given a set of tasks (test sets for the cores), a set of test resources (e.g., test buses, BIST hardware) and a test access architecture, we determine start times for the tasks such that the total test application time is minimized. We show that the test scheduling decision problem is equivalent to the m-processor open shop scheduling problem and is therefore NP-complete. However, a commonly encountered instance of this problem ($m = 2$) can be solved in polynomial time. For the general case ($m > 2$), we present a mixed-integer linear programming (MILP) model for optimal scheduling and apply it to a representative core-based system using an MILP solver available in the public domain. We also extend the MILP model to allow optimal test set selection from a set of alternatives. Finally, we present an efficient heuristic algorithm for handling larger systems for which the MILP model may be infeasible.**

***Index Terms*—Embedded core testing, open shop scheduling, resource constraints, system-on-a-chip test, test set selection, testing time.**

I. Introduction

EMBEDDED cores are increasingly being used in large system-on-a-chip (SOC) designs [17]. In order to reduce cost, the testing time for a core-based system must be minimized by carefully scheduling tests for cores at the system level, and by designing an appropriate test access architecture. Most of the previous research on core testing at the system level has focussed on the latter problem, i.e., the design of efficient test access architectures [5], [6], [10], [13], [15]. Test scheduling for core-based systems has received much less attention. Given a set of tasks (test sets for the cores), a set of test resources such as test buses and built-in self-test (BIST) logic, and a test access architecture, test scheduling refers to the problem of determining start times for the tasks such that the total test application time is minimized.

A number of scenarios are possible for core testing at the system level. The embedded cores in an SOC may be tested using BIST, external testing, or a combination of the two methods. BIST offers at-speed test capability and is necessary for detecting performance-related defects and nonmodeled faults. However, it may be necessary to augment BIST with external testing in order to detect modeled faults that are random-pattern-resistant. For external testing, the test buses that are used for test access may be shared among multiple cores. If BIST is used, then a core may either be "BIST-ed," in which case it has dedicated BIST logic, or it may simply be "BIST-ready" without containing BIST pattern generators and response monitors. In the latter case, the core user (system integrator) may design BIST logic that is shared by multiple cores. In order to minimize the testing time, the test resources in the system (test buses, BIST logic) should be carefully allocated to the various cores, and the tests for the cores should be optimally scheduled.

Sugihara *et al.* [16] recently addressed the problem of selecting a test set for each core from a set of test sets provided by the core vendor and scheduling these tests in order to minimize the testing time. (If a core is not BIST-ed, the core vendor may provide external test sets augmenting various BIST pseudorandom test lengths.) Each test set consists of a subset of patterns for BIST (implicitly denoted by the pseudorandom test length if the core is not BIST-ed) and a subset of patterns for external testing. This requires the core vendor to provide multiple test sets for each core, with the test sets containing varying proportions of patterns for BIST and external testing. Test scheduling is formulated in [16] as a combinatorial optimization problem, which is then solved using a heuristic method. The authors make two restrictive assumptions 1) every core has its own BIST logic, i.e., the BIST components of the test sets for any two cores can be assigned identical starting times, and 2) external testing can be carried out for only one core at a time, i.e., there is only one test access bus at the system level.

We formulate a generalized test scheduling problem that includes the problem addressed in [16] as a special case. We assume that the test access architecture has been predetermined, and the cores have been assigned to test buses. No restrictions are placed either on the sharing of BIST logic between cores or on the use of multiple test buses for external testing. We address fundamental computational complexity issues for test scheduling and develop a mixed-integer linear programming (MILP) model for solving the scheduling problem. Our model is easily able to handle the case where a test set for a core has to be selected from a number of alternatives provided by the core vendor. The main contributions of the paper are summarized below.

- We relate the general problem of test scheduling for SOC's to the open shop scheduling problem [9], [11], and thereby show that the test scheduling decision problem is NP-complete.

Manuscript received November 12, 1999; revised March 1, 2000 and May 25, 2000. This work was supported in part by the National Science Foundation (NSF) under Grant CCR-9875324, in part by a contract from Delphi Delco Electronics Systems, and in part by an equipment grant from Sun Microsystems. A preliminary version of this paper appeared in *Proc. Int. Conf. Computer-Aided Design*, pp. 391–394, San Jose, CA, Nov. 1999. This paper was recommended by Associate Editor R. Aitken.

The author is with the Department of Electrical and Computer Engineering, Duke University, 130 Hudson Hall, Box 90291, Durham, NC 27708 USA (e-mail: krish@ee.duke.edu).

Publisher Item Identifier S 0278-0070(00)09150-8.

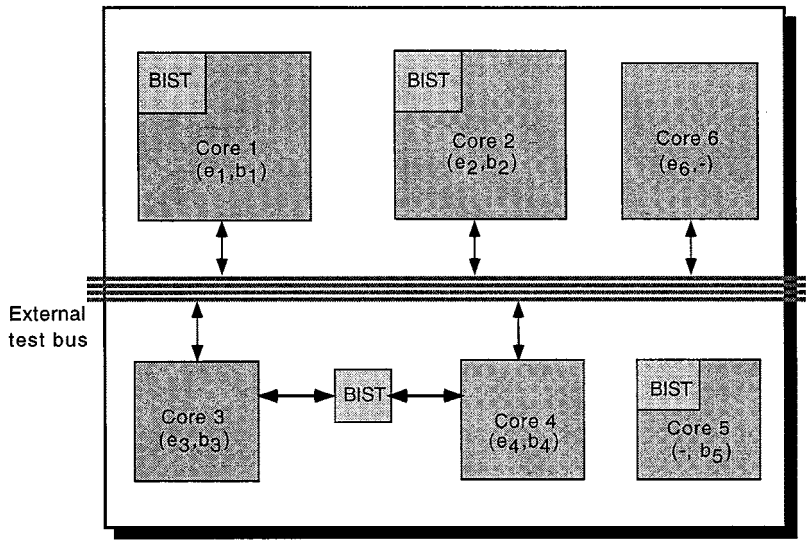


Fig. 1. An example of a generic core-based system with one external test bus, and shared and dedicated BIST logic for the cores.

- We relate a special case of the scheduling problem to open shop scheduling with two processors, and present a polynomial-time algorithm for it.
- Even though the scheduling problem is NP-complete, we show that it can be solved exactly for realistic core-based systems using MILP. The optimization problem is initially formulated as a nonlinear mixed-integer program. We then use linearization methods to obtain an MILP model.
- We evaluate the feasibility of the proposed MILP models by solving them using a linear programming solver for a representative core-based system. The experimental results demonstrate that optimal solutions to these important design problems in SOC testing are indeed feasible.
- In order to efficiently solve the scheduling problem for larger systems for which the MILP approach takes excessive time, we present a heuristic "shortest-job-first" algorithm.

The proposed test scheduling approach is best suited to hard (IP) cores whose test sets (BIST and external) are determined by the core vendor. However, it can also be applied to synthesizable cores as long their tests are not modified after the cores are inserted in the SOC and the overall SOC test schedule is determined.

The organization of the paper is as follows. In Section II, we study a special case of the test scheduling problem in which a single BIST resource and a single test bus are shared by all the cores. This arises when the core providers do not incorporate any BIST features and the system integrator, in addition to using a test bus, adds BIST to the SOC. We show that this instance of the scheduling problem is equivalent to the problem of open shop scheduling with two processors, and present an optimal, polynomial-time algorithm for it. We also present a greedy, "one-at-a-time" scheduling algorithm to demonstrate that significant savings in testing time can be achieved using optimal test schedules. For high-volume production, a small saving in testing time can translate to enormous cost savings.

In Section III, we show that the test scheduling decision problem is NP-complete by relating it to a general instance of the open shop scheduling problem. We then review MILP and develop MILP models for minimizing the testing time. We solve the MILP model for a nontrivial core-based system using the *lpsolve* software package from Eindhoven University of Technology in the Netherlands [2]. We also present a heuristic test scheduling algorithm that can easily handle larger systems for which the MILP model may not be computationally feasible. In Section IV, we extend the MILP model to include the problem of selecting a test set for each core from a set of alternatives provided by the core vendor.

A generic example of a core-based system that we consider is shown in Fig. 1. It consists of one external test bus and six cores. For core i, we assume that external test application takes e_i cycles and BIST takes b_i cycles. Core 5 is tested entirely using BIST, while Core 6 is tested entirely using external patterns. Note that in this generic example, Cores 1, 2, and 5 have dedicated BIST circuitry while Cores 3 and 4 share BIST logic.

II. Polynomial-Time Algorithm for Test Scheduling

In this section, we consider the special case in which the core-based system has only one external bus and BIST logic is shared by all the cores. As discussed in Section I, this arises when the core providers do not incorporate any BIST features and the system integrator, in addition to using a test bus, adds BIST to the SOC. The test set for every core includes BIST and external test components. Fig. 2 illustrates a system with four cores; the test lengths for external testing and BIST are also shown. For example, Core 1 requires 125 cycles for external testing and 100 cycles for BIST.

We first show that the test scheduling problem for core-based systems is equivalent to the open shop scheduling problem [9]. In the open shop scheduling problem, we are given a *shop* consisting of m processors, a set J of jobs, each job $j \in J$ consisting of m tasks $t_1[j], t_2[j], \ldots, t_m[j]$ (task $t_i[j]$ must be executed on processor i), and a length $l(t) \geq 0$ for each task. Each processor can execute only one task at a time. A schedule for an m-shop is a set of m processor schedules, one for each

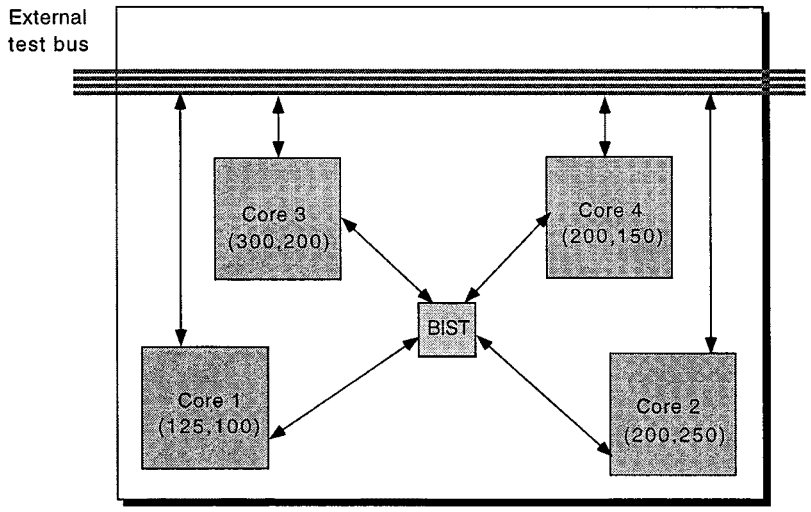


Fig. 2. An example of a core-based system with one external test bus, and BIST logic shared by all the cores.

```
Procedure SCHEDULE({e_i, b_i})
begin
  T_1 := 0; T_2 := 0; e_0 := 0; b_0 := 0; l := 0; r := 0; S := φ;
  /* e_0 and b_0 are dummy variables and the pair represents a dummy job */
  /* T_1: sum of task times for external tests */
  /* T_2: sum of task times for BIST tests */
  /* l: index of leftmost job */
  /* r: index of rightmost job */
  /* S_1: sequence of tasks for external test */
  /* S_2: sequence of tasks for BIST */
  for i:=1 to n do /* system contains n cores */
   begin
     T_1 := T_1 + e_i; T_2 := T_2 + b_i;
     if e_i ≥ b_i then
        if e_i ≥ b_r then /* Put r on right */
           S := S + r; r := i;
        else /* Put i on right */
           S := S + i;
     else
        if b_i ≥ e_l then /* Put l on left */
           S := l + S; l := i;
        else S := i + S; /* Put i on left */
   end
/* Finishing touches */
if T_1 − e_l < T_2 − b_r then
  S_1 := S + r + l; S_2 := l + S + r;
else
  S_1 := l + S + r; S_2 := r + l + S;
delete_zeros(); /* Delete all occurrences of zeros from S_1 and S_2 */
end
```

Fig. 3. An optimal test scheduling algorithm for a system with one external test bus and one shared BIST resource.

processor in the shop. These m processor schedules must be such that no job is processed simultaneously on more than one processor. The *finish time* of a schedule is the latest completion time of the individual processor schedules. The objective in open shop scheduling is to minimize the finish time.

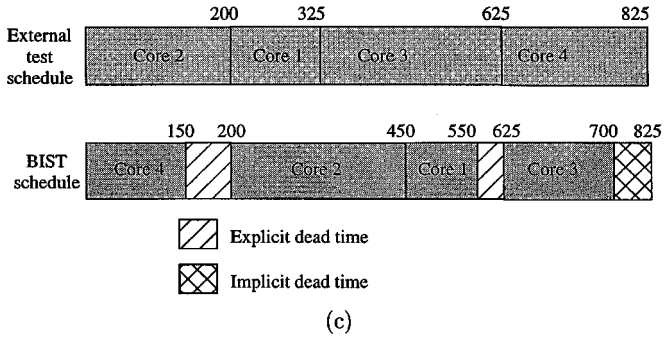
In order to establish equivalence between test scheduling for core-based systems and open shop scheduling, we view the test sets for the cores as jobs. Each job consists of two tasks, corresponding to the external test and BIST components of the test set, respectively. In the instance of the problem being considered in this section, $m = 2$, i.e., there are two processors in the system, corresponding to the external test bus and the BIST resource, respectively. An optimal schedule, i.e., one with the least finish time, guarantees the shortest testing time for

Job (core) i	Task e_i	Task b_i
1	125	100
2	200	250
3	300	200
4	200	150

(a)

End of iteration	S	l	r	T_1	T_2	Comments
0	ϕ	0	0	0	0	Initial values
1	0	0	1	125	100	$e_1 > b_1$ and $e_1 > b_r \Rightarrow$ $S := S + 0$, $r := 1$
2	00	2	1	325	350	$b_2 > e_2$ and $b_2 > e_l \Rightarrow$ $S := 0 + S$, $l := 2$
3	001	2	3	625	550	$e_3 > b_3$ and $e_3 > b_r \Rightarrow$ $S := S + 1$, $r := 3$
4	0013	2	4	825	700	$e_4 > b_4$ and $e_4 \geq b_3 \Rightarrow$ $S := S + 3$, $r = 4$
Finishing touches	$T_1 - e_2 > T_2 - b_4 \Rightarrow$ $S_1 := 2 + S + 4$, i.e. $S_1 := 2134$ (delete zeros), $S_2 := 4 + 2 + S$, i.e. $S_2 := 4213$ (delete zeros)					

(b)



(c)

Fig. 4. An example of the execution of Procedure SCHEDULE: (a) Test data (b) Iterations of SCHEDULE (c) An optimal schedule.

the core-based system. We can obtain efficient test schedules by exploiting the fact that the 2-shop scheduling problem can be solved efficiently using an $O(n)$ algorithm, where n is the number of jobs (cores) [11]. Fig. 3 provides a pseudocode description of this algorithm. The external test schedule is denoted by the list S_1, while S_2 denotes the schedule for BIST. The symbol "+" is used to denote the concatenation operation.

The algorithm proceeds by dividing the jobs into two groups, say A and B [11], and it is adapted for test scheduling as follows. The jobs in A have $c_i \geq b_i$ while those in B have $c_i < b_i$. The schedule is built from the "middle" with jobs from A added on at the right and those from B added on at the left. (S denotes a "composite schedule" which is used later to derive schedules for external testing and BIST.) Finally, some finishing touches involving only the leftmost and rightmost jobs (pointed to by l and r, respectively) are made, and this guarantees an optimal schedule [11].

Fig. 4 illustrates the execution of the procedure for the core-based system in Fig. 2. In the first iteration, since c_1 (125) exceeds both b_1 (100) and b_0 (0), we (implicitly) add job 1, i.e., core 1, from A to the right of S ($r = 1$). In the second iteration, b_2 (200) exceeds both c_2 (200) and b_0 (0), hence job 2 from B gets (implicitly) added to the left of S ($l = 2$). Next, in the third iteration, the first job is (explicitly) added to the right of S. This process continues until all four iterations are completed. Finally, the 0s in S are deleted and the tasks denoted by l and r are added to the BIST and external test schedules.

The schedules for external test and BIST are given by S_1: 2134 and S_2: 4213, respectively, and the optimal testing time for this system is 825 cycles; see Fig. 4(c). Note that the BIST schedule in Fig. 4 contains idle times. We define two types of idle times: 1) *explicit dead time*, which arises due to resource conflicts, and 2) *implicit dead time*, which may occur either at the beginning or at the end of a schedule and is not caused by resource conflicts. The explicit and implicit dead times are marked on the BIST schedule in Fig. 4(c).

A nonoptimal schedule using a greedy "one-at-a-time" scheduling algorithm is shown in Fig. 5. We assume that a fixed ordering is imposed on the cores and that the tests for the cores can be scheduled in this order with the start times determined by the availability of resources. This algorithm is therefore similar to a priority-based scheduling algorithm.

The schedule shown in Fig. 5, was generated using the ordering (1, 2, 3, 4) with the additional restriction that the ith test set for the external test schedule is determined before the ith test set for the BIST schedule. The algorithm proceeds by first scheduling the external test set for Core 1. It then schedules the BIST test set for Core 2 since the BIST test set for Core 1 cannot be scheduled at this time. Next, it successfully schedules the external test set for Core 3. (The external test set for Core 2 cannot be scheduled before its BIST test session completes.). Proceeding in this fashion, we generate a complete schedule, noting that an explicit dead time of 75 cycles must be inserted in the external test schedule after Core 2 is tested. This increase the SOC testing time to 900 cycles.

As another example, we consider a representative core-based system $\mathcal{S}$ consisting of six embedded cores from the ISCAS benchmarks [3], [4]. These benchmark circuits are known to contain random-pattern-resistant faults, hence we use pseudo-random patterns for BIST as well as deterministic patterns (applied externally) for the hard faults. Table I presents the test data for each embedded core in this system. We assume that the s5378 circuit contains four internal scan chains, while each of the s1196 and s953 circuits contain a single internal scan chain. We also assume without loss of generality that a 32-bit external test bus is used. Finally, we use the parameter *test width* $\phi_i = \max\{n_i, m_i\}$, where n_i (m_i) equals the number of inputs (outputs) in core i, to determine the external testing time for each core.

The (external) testing time for core i is determined by its test width and the width of the test bus. Let t_i be the number of (scan) patterns for core i. Let T_i be the number of external test cycles required by core i. If $\phi_i > 32$, then the width of the test bus is insufficient for parallel loading of test data, and serialization of the test data is necessary within the wrapper at the inputs and/or outputs of core i. In order to calculate the test time due to serialization, we assume the interconnection strategy suggested in [13] and used in [5], [6] for connecting core input–outputs (I/Os) to the test bus, namely, provide direct (parallel) connection to core I/Os that transport more test data (Fig. 6). This

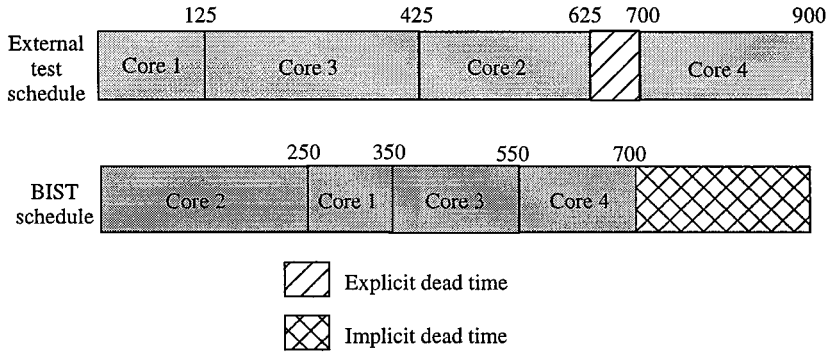


Fig. 5. A greedy, nonoptimal schedule generated using a one-at-a-time strategy.

TABLE I
TEST DATA FOR THE CORES IN SYSTEM $\mathcal{S}$

Circuit (core)	i	Number of test inputs n_i	Number of test outputs m	$\phi_i = \max\{n_i, m_i\}$	Number of external test patterns p_i	Number of scan test patterns t_i	Number of external test cycles	Number of BIST patterns	Number of BIST cycles
c880	1	60	26	60	13	13	377	4096	4096
c2670	2	233	140	233	79	79	15958	64000	64000
c7552	3	207	108	207	48	48	8448	64000	64000
s953	4	45	52	52	45	1379	28959	4096	217140
s5378	5	39	53	53	59	2759	60698	4096	389214
s1196	6	32	32	32	40	778	778	4096	135200

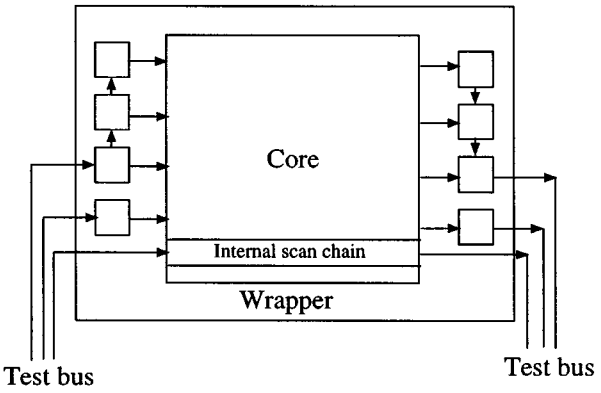


Fig. 6. Illustration of the serialization model.

strategy is effective for cores that have a small number of scan I/Os but whose scan chains are long compared to the number of functional I/Os. Note that the proposed SOC test scheduling approach does not depend on the serialization strategy, hence alternative serialization models can likewise be considered for calculating the testing time for the individual cores. These testing times can then be used as inputs to the test scheduling framework.

If the width of bus j is adequate, i.e., $\phi_i \leq 32$ (for a 32-bit test bus), then no serialization is necessary and core i can be tested in exactly t_i cycles. It can be easily seen that

$$T_i = \begin{cases} t_i, & \text{if } \phi_i \leq 32 \\ (\phi_i - 31)t_i, & \text{if } \phi_i > 32. \end{cases}$$

For example, if we refer to the core in Fig. 6 as core i, then $\phi_i = 5$ and the test bus is three bits wide. Hence $T_i = (5-3+1)t_i = 3t_i$.

For the combinational cores, $1 \leq i \leq 3$, the number of scan test cycles t_i is equal to the number of test patterns p_i. However, for the remaining three cores with internal scan, $t_i = (p_i+1)\lceil f_i/N_i \rceil + p_i$, where core i contains f_i flip-flops and N_i internal scan chains [1]. The test patterns for all these circuits were obtained by applying the Atalanta ATPG program [12] to the hard faults.

Fig. 7 illustrates the scheduling algorithm for the core-based system of Table I. We assume that the application of a BIST pattern takes one clock cycle and external test application is ten time slower than BIST pattern application. The values of T_1 and T_2 are shown as multiples of 10. For this example, the two lists are S_1: 612 345 and S_2: 561234, respectively. This yields the schedule shown in Fig. 7(c).

III. TEST SCHEDULING: GENERAL CASE

While the special case of the scheduling problem discussed in Section II can be easily solved using a linear-time algorithm, the general case of $m > 2$ corresponding to more than one BIST resource in the system is NP-complete [11]. In this section, we develop MILP models to solve the test scheduling problem.

We first briefly review mixed-integer linear programming using matrix notation [14]. The goal of MILP is to minimize a linear objective function on a set of integer and/or real variables, while satisfying a set of linear constraints. A typical MILP model is described as follows:

$$\begin{aligned} \text{minimize:} \quad & \mathbf{Ax} + \mathbf{By} \\ \text{subject to:} \quad & \mathbf{Cx} + \mathbf{Dy} \leq \mathbf{E}, \mathbf{x} \geq 0, \mathbf{y} \geq 0 \end{aligned}$$

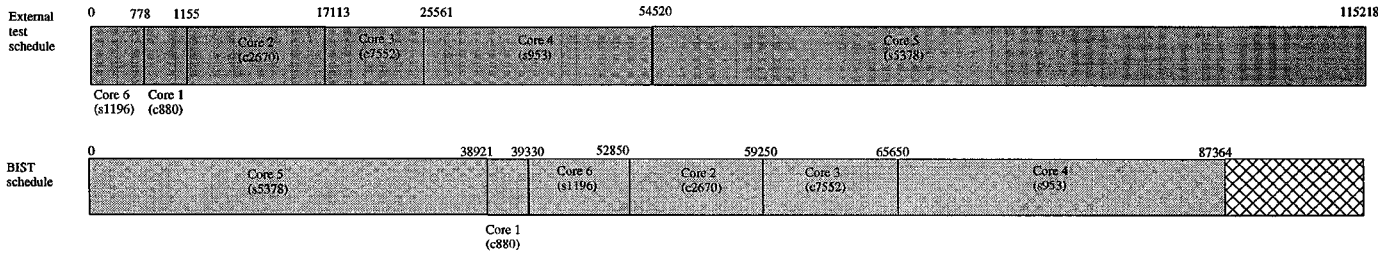
where $\mathbf{A}$ and $\mathbf{B}$ are cost vectors, $\mathbf{C}$ and $\mathbf{D}$ are constraint matrices, $\mathbf{E}$ is a column vector of constants, $\mathbf{x}$ is a vector of integer variables, and $\mathbf{y}$ is a vector of real variables. The MILP models presented in this paper were derived after extensive experimentation, and they involve the linearization of a number of nonlinear constraints. Efficient MILP solvers are now readily available, both commercially and in the public domain. For our

Job (core) i	Task e_i	Task b_i
1	377	409
2	15958	6400
3	8448	6400
4	28959	21714
5	60698	38921
6	778	13520

(a)

End of iteration	S	l	r	T_1	T_2	Comments
0	ϕ	0	0	0	0	Initial values
1	0	1	0	377	409	$e_1 < b_1$ and $b_1 > e_l \Rightarrow$ $S := 0 + S$, $l := 1$
2	00	1	2	16335	6809	$e_2 > b_2$ and $e_2 > b_r \Rightarrow$ $S := S + 0$, $r := 2$
3	002	1	3	24783	13209	$e_3 > b_3$ and $e_3 > b_r \Rightarrow$ $S := S + 2$, $r := 3$
4	0023	1	4	53742	34923	$e_4 > b_4$ and $e_4 > b_3 \Rightarrow$ $S := S + 3$, $r = 4$
5	00234	1	5	114441	73844	$e_5 > b_5$ and $e_5 > b_r \Rightarrow$ $S := S + 4$, $r = 5$
6	100234	6	5	115218	87364	$e_6 < b_6$ and $b_6 > e_1 \Rightarrow$ $S := 1 + S$, $l = 6$
Finishing touches	$T_1 - e_6 > T_2 - b_5 \Rightarrow$ $S_1 := 6 + S + 4$, i.e. $S_1 := 612345$ (delete zeros), $S_2 := 5 + 6 + S$, i.e. $S_2 := 561234$ (delete zeros)					

(b)



(c)

Fig. 7. (a) Test times (tasks) for the cores in System $\mathcal{S}$ (b) Execution of Procedure SCHEDULE (c) An optimal schedule.

experiments, we used the *lpsolve* package from Eindhoven University of Technology in the Netherlands [2].

We now address the test scheduling problem for a given core-based system. In order to minimize the testing time, the start times of the external and BIST test sets must be optimally determined. Let $T = \{t_1, t_2, \ldots, t_{2N_C}\}$ denote the start times of the set of test patterns (external and BIST) that must be applied to the cores in the system. The start time of the external test set e_i for core i is denoted by t_{2i-1} while the start time of the BIST test set b_i for core i is denoted by t_{2i}. For notational convenience, we will also refer to these test sets by the subscripts of the variables that denote their start times. For example, we will interchangeably use e_i and $2i-1$ to refer to the external component of the test set of core i.[1]

[1]Note that e_i and $2i-1$ refer to the external test set for core i while t_{2i-1} refers to the start time for e_i. Similarly, b_i and $2i$ refer to the BIST test set for core i while t_{2i} refers to the start time for b_i.

Let $L = \{l_1, l_2, \ldots, l_{2N_C}\}$ denote the corresponding test lengths (number of cycles) for the test sets. Note that if the test set for core i has no external test (BIST) component, then $t_{2i-1} = 0$ and $l_{2i-1} = 0$ ($t_{2i} = 0$ and $l_{2i} = 0$). Two test sets i and j *overlap* if either 1) $t_i < t_j + l_j$ and $t_i + l_i > t_j$, or 2) $t_j < t_i + l_i$ and $t_j + l_j > t_i$. If there is only one external test bus, the external test components for the cores in any valid test schedule must not overlap; therefore, a lower bound on the system testing time in this case is given by $\sum_{j=1}^{N_C} e_j$. Another lower bound on the testing time is given by $\max_j\{e_j + b_j\}$. Note also that test sets i and j do not overlap if and only if either 1) $t_i - t_j - l_j \geq 0$, or 2) $t_j - t_i - l_i \geq 0$.

Two test sets are *conflicting* if they cannot be applied to the system at the same time. Test sets can be conflicting if 1) they share an external test bus, or 2) they are BIST test sets for cores that share a BIST resource, or 3) they are the external and BIST components of a core's test set. Clearly, there cannot be any overlap between conflicting test sets.

Minimize C subject to:

1. $C \geq t_i + l_i$, $1 \leq i \leq 2N_C$
2. $x_{ij}(s_{ij1} - s_{ij2} - l_j\delta_{ij1}) + x_{ij}(s_{ij3} - s_{ij4} - l_i\delta_{ij2}) \geq 0$, $1 \leq i, j \leq 2N_C$
3. $\delta_{ij1} + \delta_{ij2} = 1$, $1 \leq i, j \leq 2N_C$
4. $s_{ij1} - M\delta_{ij1} \leq 0$, $M = \sum_{i=1}^{N_C} l_i$, $1 \leq i, j \leq 2N_C$
5. $-t_i + s_{ij1} \leq 0$, $1 \leq i, j \leq 2N_C$
6. $t_i - s_{ij1} + M\delta_{ij1} \leq M$, $1 \leq i, j \leq 2N_C$
7. $\delta_{ij1}, \delta_{ij2} = 0$ or 1, $1 \leq i, j \leq 2N_C$.

Fig. 8. MILP model for $\mathcal{P}1$.

The optimization problem that we address in this section is to minimize the system testing time by optimally determining the start times $t_1, t_2, \ldots, t_{2N_C}$ for the various test sets. The formal statement of the problem is as follows.

- $\mathcal{P}1$: Given a system with N_C cores such that core i ($1 \leq i \leq N_C$) has BIST test length $l_{2i} \geq 0$ and external test length $l_{2i-1} \geq 0$, determine the start times $t_1, t_2, \ldots, t_{2N_C}$ for the BIST and external test sets such that 1) conflicting test sets do not overlap, and 2) the overall system testing time, i.e., $\max_i\{t_i + l_i\}$, is minimized.

Let x_{ij}, $1 \leq i, j \leq 2N_C$, be a 0-1 variable defined as follows:

$$x_{ij} = \begin{cases} 1, & \text{if the test sets } i \text{ and } j \text{ are conflicting} \\ 0, & \text{otherwise.} \end{cases}$$

We now develop an MILP model for $\mathcal{P}1$. We first formulate the model in terms of nonlinear constraints, and then linearize it using standard techniques.

Objective: Minimize the cost function

$$C = \max_{i \in \{1,2,\ldots 2N_C\}} \{t_i + l_i\} \quad \textbf{subject to :}$$
$$x_{ij}(t_i - t_j - l_j) \geq 0 \ \textbf{or} \ x_{ij}(t_j - t_i - l_i) \geq 0, \leq i, j \leq 2N_C.$$

The above minmax nonlinear cost function can easily be linearized [14] by minimizing the (real) variable C and adding the constraints $C \geq t_i + l_i$, $1 \leq i \leq 2N_C$. However, it is more difficult to linearize the nonlinear constraints containing the logical **or** construct. 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 problem is now restated as:

Objective: Minimize the cost function C **subject to**:

1) $C \geq t_i + l_i$, $1 \leq i \leq 2N_C$;
2) $x_{ij}\delta_{ij1}(t_i - t_j - l_j) + x_{ij}\delta_{ij2}(t_j - t_i - l_i) \geq 0$, $1 \leq i, j \leq 2N_C$;
3) $\delta_{ij1} + \delta_{ij2} = 1$, $1 \leq i, j \leq 2N_C$;
4) $\delta_{ij1}, \delta_{ij2} = 0$ or 1, $1 \leq i, j \leq 2N_C$.

The constraint 2) above is still nonlinear. We linearize it by replacing $t_i\delta_{ij1}$ by the (real) variable s_{ij1} and $t_j\delta_{ij1}$ by the (real) variable s_{ij2}. Similarly, we replace $t_j\delta_{ij2}$ by s_{ij3} and t_ix_{ij2} by s_{ij4}. For each such substitution, we add three additional constraints. For example, for the substitution of $t_i\delta_{ij1}$ by s_{ij1}, we add the following constraints:

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

where $M = \sum_{i=1}^{N_C} l_i$ is an upper bound on the value of t_i, $1 \leq i \leq 2N_C$. The resulting MILP model is shown in Fig. 8.

We applied the MILP model of Fig. 8 to the core-based system described in Table I for several test scenarios corresponding to varying amount of on-chip BIST resources. The complexity of the MILP model depends on the number of cores and the test resource conflicts, and is independent of the sizes of the cores. As in Section II, we assumed that the application of a BIST pattern takes one clock cycle and external test application is ten time slower than BIST pattern application. We solved the MILP models using the *lpsolve* software package on a Sun Ultra 10 workstation with a 300-MHz processor and 128 MB memory. We were unable to obtain actual CPU times from *lpsolve*; however, the user time was less than one minute in each case. The optimum testing time for this system is 1152810 cycles. Fig. 9 shows optimal schedules when the cores share BIST logic. The explicit and implicit dead times for these optimal schedules are also shown.

It follows therefore that the lower bound of $\sum_{i=1}^{N_C} e_i$ for the system testing time is achieved only if the external test schedule has no dead time. This is indeed the case in the schedule shown in Figs. 4 and 9. However, we next show that this lower bound is not always achieved, even with an optimal schedule.

Consider a smaller example consisting of four cores, namely c7552, s953, s1196, and s5378. Assuming that each core has its dedicated BIST logic and using the test data listed in Fig. 7(a), we obtain an optimal test schedule shown in Fig. 10. The lower bound for this system obtained from the external testing times is 981 050 cycles; however, the optimum testing time is 996 190 cycles, the difference between these two figures is due to explicit dead time in the external test schedule. The dead time can be eliminated if the BIST patterns for s5378 are applied in two test sessions. However, such *preemptive scheduling* can complicate the test controller and is therefore not considered in this paper.

We next develop an optimal test schedule for the system of Table I when an additional core (the s13297 ISCAS 89 benchmark circuit) that is tested entirely using BIST is added to it. The s13207 circuit is known to be random-pattern-testable. We assume that 512K random patterns are applied to it in a test-per-clock fashion, hence BIST for this circuit is assumed to take 512K cycles. The optimum testing time for this system is 1 182 350 cycles, and an optimal schedule is shown in Fig. 11. For this example, the optimum testing time is determined by the BIST components of the core test sets. The external test schedule contains both explicit and implicit dead times.

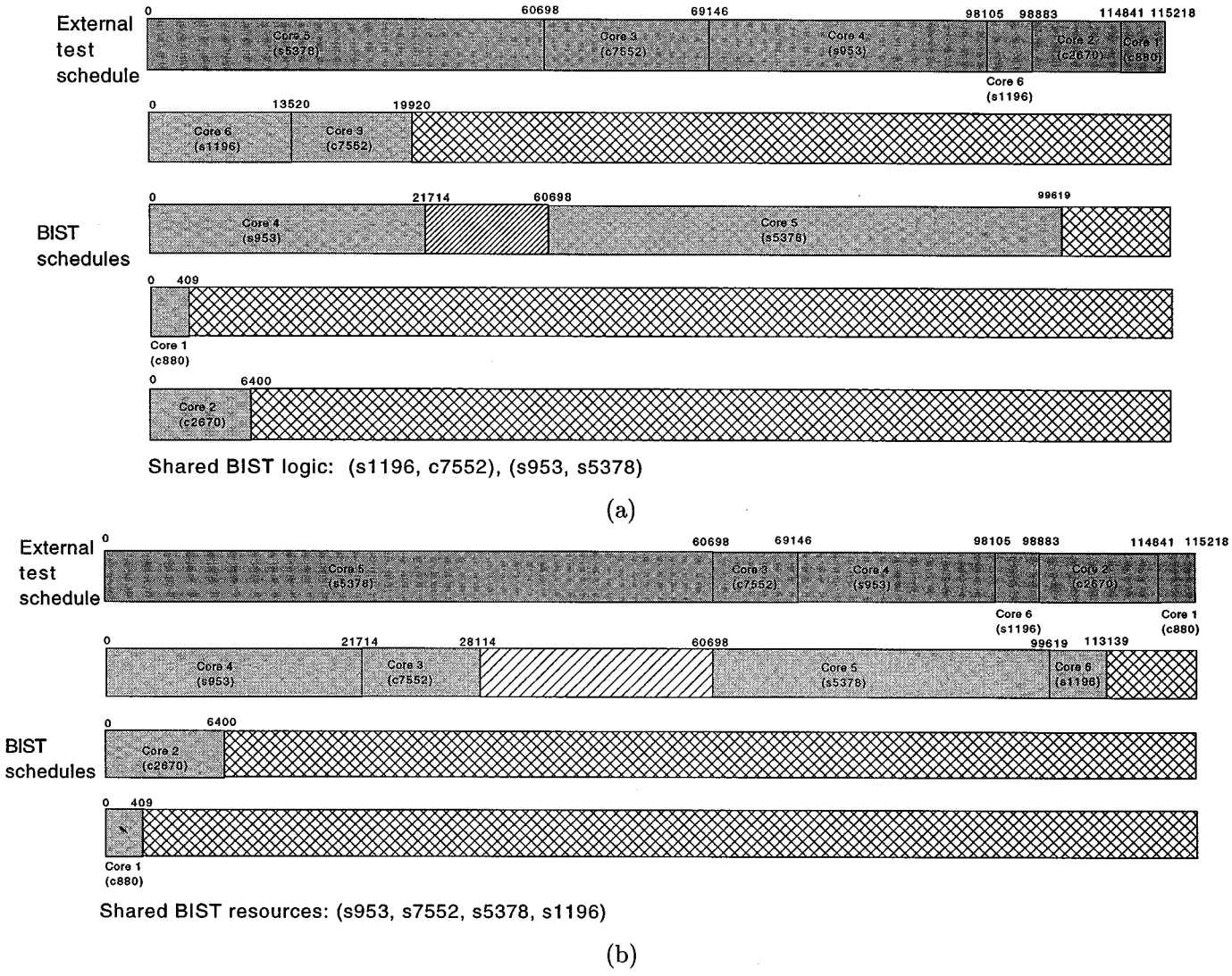


Fig. 9. Test schedules for the core-based system of Table I when cores share BIST resources: (a) BIST logic is shared between s1196 and c7552, and between s953 and s5378; (b) BIST logic is shared between s953, c7552, s5378, and s1196.

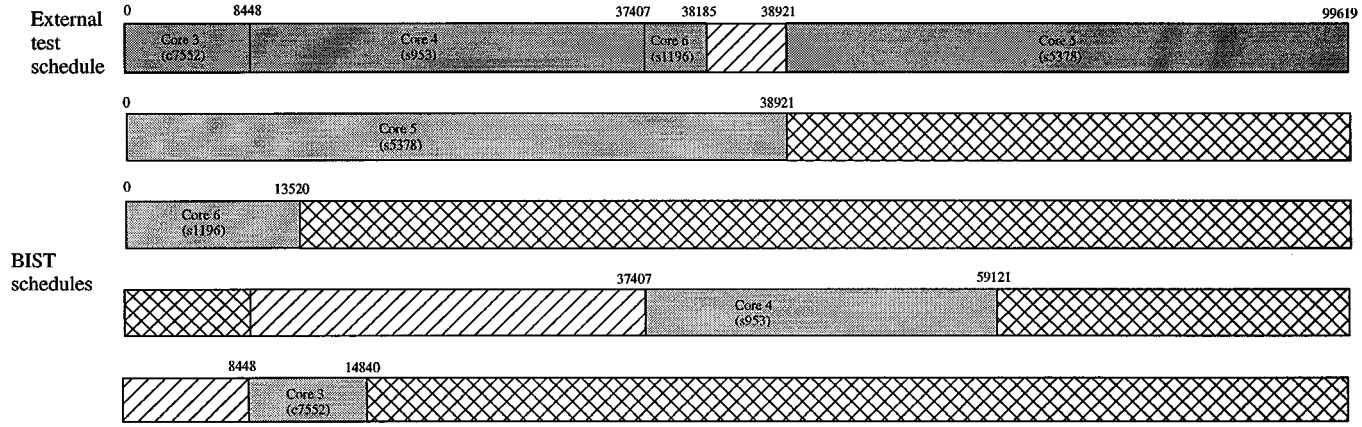


Fig. 10. An example showing that an optimal test schedule may contain explicit dead times for external testing.

Since the test scheduling problem is NP-complete, the amount of time required to generate and solve the ILP models for significantly larger core-based systems may be excessive. In order to handle such systems, we now present a shortest-task-first heuristic scheduling algorithm. This algorithm first initializes the start times of the test sets (tasks) to zero. It then

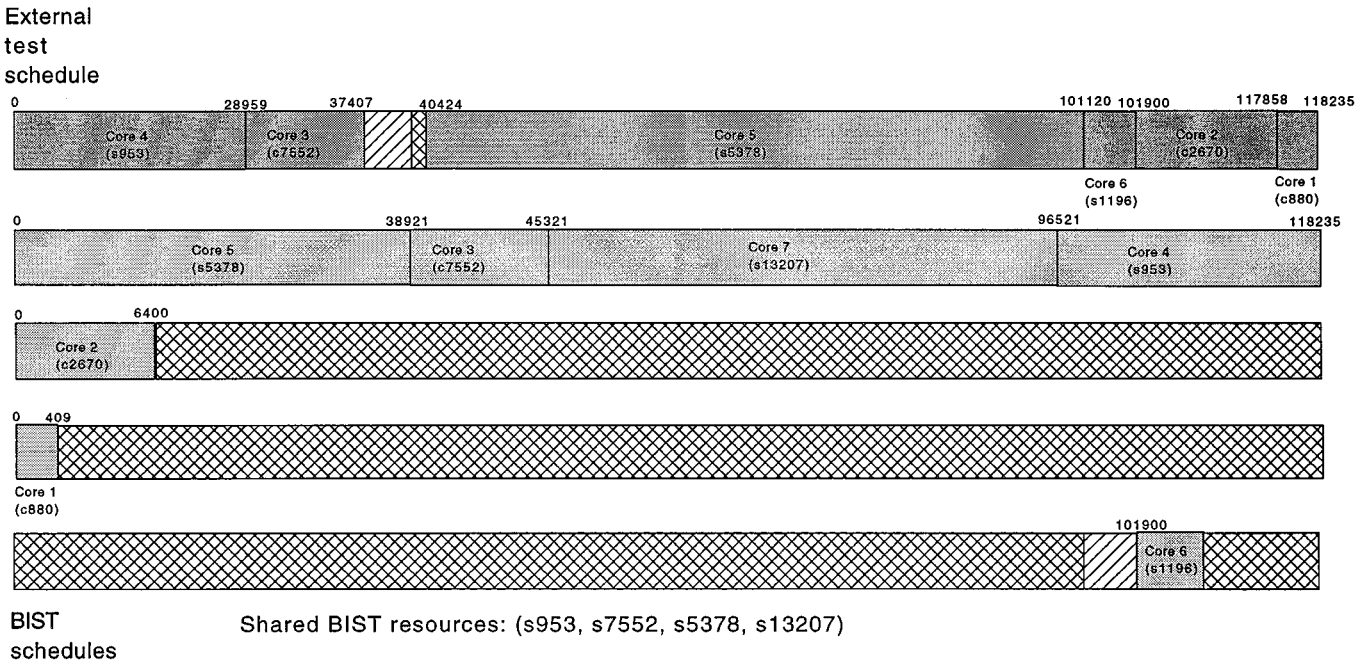


Fig. 11. An optimal test schedule for the system with seven cores.

```
Procedure SHORTEST-TASK-FIRST({t_i})
begin
for i:=1 to r do /* there are r tasks */
   start_time_i := 0; /* Initialize all start times to zero */
while flag = 1 do /* Indicates resource conflict */
   begin
   flag = 0;
   for i:=1 to r do
      for j:=i+ 1 to r do
         if x_ij = 1 then /* x_ij = 1 if i and j are conflicting */
            if OVERLAP(i, j) then
      /* Execution of tasks i and j overlap even though they share a test resource */
                  begin
                  if start_time_i + l_i ≥ start_time_j + l_j then /* If j completes before i */
                     start_time_i := start_time_j + l_j; /* Start i after j completes */
                  else start_time_j := start_time_i + l_i;
                  flag = 1; end
   end
end
```

Fig. 12. The shortest-task-first procedure.

iteratively checks for a valid schedule by examining all pairs of test sets that are conflicting, i.e., they share test resources and overlap. If a conflict is detected, i.e., OVERLAP$(i, j) = 1$ and $x_{ij} = 1$, the task that completes later is re-scheduled such that it starts after the completion of the task with the earlier completion time. In this way, the procedure schedules shorter tasks first and iteratively updates the start times until all resource conflicts are eliminated.

A pseudocode description of this algorithm is provided in Fig. 12. The worst-case time complexity of the algorithm is $O(r^3)$, where r is the number of tasks. This can be explained as follows. In each iteration of the **while** loop, at most r^2 operations are performed corresponding to the two **for** loops. Also, in iteration i of the **while** loop, the start times of at most $r - i$ tasks are updated. This implies that the start times settle after at most r iterations of the **while** loop.

The shortest-task-first algorithm yields a testing time of 1 213 330 cycles for the system of Table I when s13207 is added, and s953, s7552, s5378, and s13207 share BIST logic. This is only 2.6% greater than the optimum testing time obtained using the MILP model. For this example, four iterations are sufficient to determine the start times (Table II). The fourth

TABLE II
START TIMES OBTAINED USING THE SHORTEST-TASK-FIRST ALGORITHM

Test set index	Test set	Iteration 1 Start time	Iteration 1 Completion time*	Iteration 2 Start time	Iteration 2 Completion time	Iteration 3 Start time	Iteration 3 Completion time
1	c2670 (BIST)	0	64000	0	64000	0	64000
2	s953 (BIST)	0	217140	0	217140	0	217140
3	s5378 (BIST)	0	389210	217140†	606350	217140	606350
4	c7552 (external)	0	84480	64000†	148480	64000	148480
5	s953 (external)	0	289590	308060†	597650	308060	597650
6	s5378 (external)	0	606980	606350†	1213330	606350	1213330
7	s1196 (BIST)	0	135200	7780†	142980	11550†	146750
8	s1196 (external)	0	7780	3770†	11550	3770	11550
9	s7552 (BIST)	0	64000	0	64000	0	64000
10	c880 (BIST)	0	4090	3770†	7860	3770	7860
11	c2670 (external)	0	159580	148480†	308060	148480	308060
12	c880 (external)	0	3770	0	3770	0	3770
13	s13207 (BIST)	0	512000	606350†	1118350	606350	1118350
Conflicts		{1,4}, {2,3}, {2,5} {2,13}, {3,6}, {3,13} {4,11}, {5,11}, {7,8} {8,12}, {10,12}		{6,7}		—	

†Updated over previous iteration

iteration is not shown in Table II since it does not update any of the start times; it is only necessary to satisfy the algorithm's termination criterion.

IV. TEST SCHEDULING WITH MULTIPLE TEST SETS

In this section, we develop an MILP model for the scheduling problem that was introduced by Sugihara *et al.* [16]. The goal here is to select test sets for the cores from a set of alternatives with a varying proportion of BIST and external test patterns, and determine their start times such that the system testing time is minimized. As described in [16], these alternatives allow the system integrator to reduce testing time by optimizing the usage of shared test resources. For example, it may be possible to test the SOC in fewer cycles if an appropriate BIST (external) test set is used which can be accomodated in an available slot in the BIST (external) test schedule.

While Sugihara *et al.* provide a heuristic solution and make the restrictive assumption that the cores do not share BIST logic, our general MILP model allows sharing of BIST resources among cores and provides an exact solution. The extension of the MILP model of Section III to this problem also demonstrates its expressive power.

Suppose we have N_i alternative test sets for core i in the system. These test sets may contain varying proportion of BIST patterns. We denote the test lengths for the BIST patterns for core i by $l_{2i,1}$, $l_{2i,2}$, ... l_{2i,N_i}. Similarly, we denote the test lengths for the external patterns for core i by $l_{2i-1,1}, l_{2i-1,2}, \ldots, l_{2i-1,N_i}$. If the kth test set is chosen for core i, then it consists of $l_{2i,k}$ cycles for BIST and $l_{2i-1,k}$ cycles for external testing.

We use the parameter x_{ij} and the variables t_i, δ_{ij1}, and δ_{ij2} ($1 \le i, j \le 2N_C$) as defined in the MILP model for $\mathcal{P}2$. In addition, we use a 0-1 indicator variable λ_{ik} ($1 \le i \le 2N_C$, $1 \le k \le N_i$), which is set to 1 if the kth test set (consisting of BIST and external test patterns) is selected for core i. The formal statement of the optimization problem is as follows.

- $\mathcal{P}2$: Given a system with N_C cores such that core i has N_i test sets, its kth test set ($1 \le k \le N_i$) has BIST test length $l_{2i,k} \ge 0$ and external test length $l_{2i-1,k} \ge 0$, select a test set for each core, and determine the start times $t_1, t_2, \ldots, t_{2N_C}$ for the BIST and external parts of the test sets such that 1) conflicting test sets do not overlap, and 2) the overall system testing time is miminized.

We now develop the MILP model for $\mathcal{P}2$.

Objective: Minimize the cost function

$$C = \max_{i \in \{1,2,\ldots,2N_C\}} \left\{ t_i + \sum_{k=1}^{N_i} \lambda_{ik} l_{ik} \right\} \quad \textbf{subject to:}$$

1) $\sum_{k=1}^{N_i} \lambda_{ik} = 1, 1 \le i \le 2N_C$;
2) If i and j refer to the same core ($i - j = 1$ and i is even), then $\lambda_{ik} - \lambda_{jk} = 0, 1 \le k \le N_i$;
3) $\sum_{k=1}^{N_i} \lambda_{ik} = 1, 1 \le i \le 2N_C$;
4) $x_{ij}\delta_{ij1}(t_i - t_j - \sum_{k=1}^{N_j} \lambda_{jk} l_{j,k}) + x_{ij}\delta_{ij2}(t_j - t_i - \sum_{k=1}^{N_i} \lambda_{ik} l_{i,k}) \ge 0$;
5) $\delta_{ij1}, \delta_{ij2} = 0$ or $1, 1 \le i, j \le 2N_C$.

Note that constraint 4) above is derived from the MILP model for $\mathcal{P}1$. Once again, the minmax nonlinear cost function can easily be linearized [14] by minimizing the (real) variable C and adding the constraints

$$C \ge t_i + \sum_{k=1}^{N_i} \lambda_{ik} l_{i,k}, \qquad 1 \le i \le 2N_C.$$

In order to linearize constraint 4) above, we replace the product of 0-1 variables $\delta_{ij1}\lambda_{jk}$ by u_{ij1k} and $\delta_{ij2}\lambda_{ik}$ by u_{ij2k}, $1 \le i, j \le 2N_C$. We also need to add the following contraints [14]:

1) $-\lambda_{jk} + u_{ij1k} \le 0, 1 \le i, j \le 2N_C, 1 \le k \le N_j$;
2) $-\delta_{ij1} + u_{ij1k} \le 0, 1 \le i, j \le 2N_C, 1 \le k \le N_j$;
3) $\lambda_{jk} + \delta_{ij1} - u_{ij1k} \le 1, 1 \le i, j \le 2N_C, 1 \le k \le N_j$;
4) $-\lambda_{ik} + u_{ij2k} \le 0, 1 \le i, j \le 2N_C, 1 \le k \le N_i$;
5) $-\delta_{ij2} + u_{ij2k} \le 0, 1 \le i, j \le 2N_C, 1 \le k \le N_i$;
6) $\lambda_{ik} + \delta_{ij2} - u_{ij2k} \le 1, 1 \le i, j \le 2N_C, 1 \le k \le N_i$.

This yields the MILP model for $\mathcal{P}2$ shown in Fig. 13.

We next apply the MILP model for $\mathcal{P}2$ to an example of consisting of four 16-bit multiplier cores used in [16]. These cores

Minimize C subject to:

1. $C \geq t_i + \sum_{k=1}^{N_i} \lambda_{ik} l_{i,k}$
2. $x_{ij}(s_{ij1} - s_{ij2} - \sum_{k=1}^{N_j} l_k u_{ijk1}) + x_{ij}(s_{ij3} - s_{ij4} - \sum_{k=1}^{N_i} l_k u_{ijk2}) \geq 0,\ 1 \leq i, j \leq 2N_C$
3. $s_{ij1} - M\delta_{ij1} \geq 0,\ M = \sum_{i=1}^{N_C} l_i,\ 1 \leq i, j \leq 2N_C$
4. $-t_i + s_{ij1} \leq 0,\ 1 \leq i, j \leq 2N_C$
5. $t_i - s_{ij1} + M\delta_{ij1} \leq M,\ 1 \leq i, j \leq 2N_C$
6. $\delta_{ij1}, \delta_{ij2} = 0$ or $1,\ 1 \leq i, j \leq 2N_C$
7. $\delta_{ij1} + \delta_{ij2} = 1,\ 1 \leq i, j \leq 2N_C$
8. If i and j refer to the same core ($i - j = 1$ and i is even), then $\lambda_{ik} - \lambda_{jk} = 0,\ 1 \leq k \leq N_i$
9. $u_{ijk1} = 0$ or $1,\ 1 \leq i, j \leq 2N_C,\ 1 \leq k \leq N_j$
10. $u_{ijk2} = 0$ or $1,\ 1 \leq i, j \leq 2N_C,\ 1 \leq k \leq N_i$
11. $-\lambda_{jk} + u_{ij1k} \leq 0,\ 1 \leq i, j \leq 2N_C,\ 1 \leq k \leq N_j$
12. $-\delta_{ij1} + u_{ij1k} \leq 0,\ 1 \leq i, j \leq 2N_C,\ 1 \leq k \leq N_j$
13. $\lambda_{jk} + \delta_{ij1} - u_{ij1k} \leq 1,\ 1 \leq i, j \leq 2N_C,\ 1 \leq k \leq N_j$
14. $-\lambda_{ik} + u_{ij2k} \leq 0,\ 1 \leq i, j \leq 2N_C,\ 1 \leq k \leq N_i$
15. $-\delta_{ij2} + u_{ij2k} \leq 0,\ 1 \leq i, j \leq 2N_C,\ 1 \leq k \leq N_i$
16. $\lambda_{ik} + \delta_{ij2} - u_{ij2k} \leq 1,\ 1 \leq i, j \leq 2N_C,\ 1 \leq k \leq N_i$

Fig. 13. MILP model for $\mathcal{P}2$.

TABLE III
TESTS DATA FOR THE FOUR MULTIPLIER CORES [16]

Core	No. of cycles (Test set 1)		No. of cycles (Test set 2)		No. of cycles (Test set 3)	
	External	BIST	External	BIST	External	BIST
1	9	235	58	10	30	55
2	27	120	19	140	10	270
3	46	20	28	68	13	360
4	84	55	68	120	53	195

	Test lengths (External, BIST)		
Core 1	(9, 235)	(58, 10)	**(30, 55)**
Core 2	(27, 120)	**(19, 140)**	(10, 270)
Core 3	**(46, 20)**	(28, 60)	(13, 360)
Core 4	**(84, 55)**	(68, 120)	(53, 195)

Fig. 14. An optimal test set selection for the multiplier cores (the selected test sets are highlighted) obtained using the MILP model for $\mathcal{P}2$.

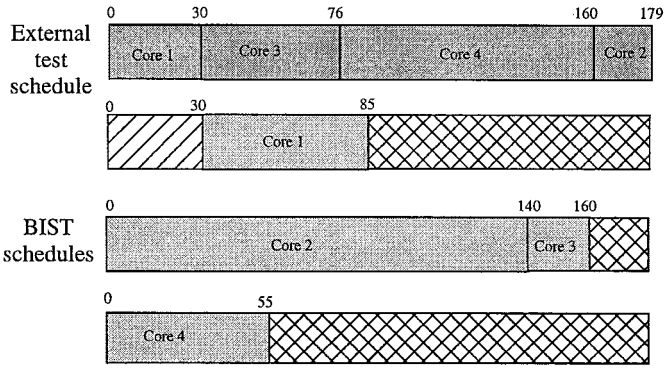


Fig. 15. An optimal schedule obtained using the MILP model for $\mathcal{P}2$ for the system described by Table III.

are described in detail in [16]; Table III presents the relevant test data. For each core, we consider three different sets of BIST and external test patterns. The heuristic approach in [16] was applied to cores with a large number of alternative test sets. However, this may be an impractical scenario—core providers may provide a few alternative test sets, but it is unrealistic to expect a large number of alternatives. Therefore, we restrict the number of alternatives to three for the case study.

We also assume that Core 2 and Core 3 share BIST logic. Fig. 14 shows an optimal selection of test sets for these cores; an optimal test schedule for this example is shown in Fig. 15. The testing time is significantly higher if alternative test sets are not provided to the system integrator. For example, if only the first pair of test sets (nine cycles for external testing and 235 cycles for BIST) is available for Core 1, the system testing time is at least 244 cycles.

V. CONCLUSION

We have presented optimal solutions to the test scheduling problem for core-based systems. Given a set of tasks (test sets for the cores), a set of test resources and a test access architecture, our scheduling methods provide start times for the tasks such that the total test application time is minimized. We have

shown that the test scheduling decision problem is equivalent to the m-processor open shop scheduling problem and is therefore NP-complete. However, a commonly encountered instance of this problem ($m = 2$) can be solved in polynomial time. This corresponds to the case in which all the cores in the system share BIST logic, e.g., when the core providers do not incorporate any BIST features and the system integrator, in addition to using a test bus, adds BIST to the SOC. For the general case ($m > 2$), we have presented a MILP model for optimal scheduling and applied it to a representative core-based system using an MILP solver available in the public domain. We have also extended the MILP model to allow optimal test set selection from a set of alternatives. In this way, our work improves upon the heuristic algorithm for test set selection described in [16]. Finally, we have presented an efficient heuristic scheduling algorithm for handling larger systems for which the MILP model may be infeasible.

Our initial results give rise to a number of new directions for further research. These are summarized below.

- The scheduling methods described in the paper make the implicit assumption that the test access architecture has been predetermined, and the cores have been assigned to test buses. We can expect more effective test schedules if the cores can be optimally assigned to test buses. This issue is currently being investigated. We are also including power constraints in the MILP model for test scheduling.
- The ILP model descriptions that we have used in our experiments are problem-specific, i.e., they are described in a format specific to the problem instance and to the *lpsolve* program. This is a cumbersome process. It is far more convenient to use high-level languages such as AMPL [7] and GAMS [8] that allow the model to be described in a *parameterized form* that is independent of the ILP solver and the input data used for a specific instance of the model.
- Significant advances have been made in recent years in solving nonlinear integer programs, and a number of these solvers are now readily available, e.g., through the Argonne National Laboratory (http://www.mcs.anl.gov/otc/Server/neos.html). We are examining the feasibility of using such nonlinear solvers for optimal test scheduling.
- Finally, we are investigating the use of preemptive scheduling for reducing the testing time further. We are also studying the shortest-time-first heuristic in more detail to determine how far it is from optimal.

ACKNOWLEDGMENT

The author would like to thank M. Sugihara of Kyushu University, and H. Date of the Institute of Systems & Information Technologies, Kyushu, Japan, for providing test data for the multiplier cores. He would also like to thank E. J. Marinissen of Philips Research Laboratories for valuable comments on an earlier version of this manuscript.

REFERENCES

[1] J. Aerts and E. J. Marinissen, "Scan chain design for test time reduction in core-based ICs," in *Proc. Int. Test Conf.*, 1998, pp. 448–457.
[2] lpsolve (version 3.0). Eindhoven Univ. Technology, Design Automation Section, Eindhoven, The Netherlands. [Online]. Available: ftp://ftp.ics.ele.tue.nl/pub/lp_solve
[3] F. Brglez and H. Fujiwara, "A neutral netlist of ten combinational benchmark circuits and a target simulator in Fortran," in *Proc. 1985 Int. Symp. Circuits and Systems*, 1985, pp. 695–698.
[4] F. Brglez, D. Bryan, and K. Kozminski, "Combinational profiles of sequential benchmark circuits," in *Proc. 1989 Int. Symp. Circuits and Systems*, 1989, pp. 1929–1934.
[5] K. Chakrabarty, "Design of system-on-a-chip test access architectures using integer linear programming," in *Proc. 2000 IEEE VLSI Test Symp.*, 2000, pp. 127–134.
[6] ——, "Design of system-on-a-chip test access architectures under place-and-route and power constraints," in *Proc. 2000 IEEE/ACM Design Automation Conf.*, pp. 432–437.
[7] R. Fourer, D. M. Gay, and B. W. Kernighan, *AMPL: A Modeling Language for Mathematical Programming*. South San Francisco, CA: Scientific, 1993.
[8] GAMS Development Corporation, *GAMS: A User's Guide*. Boston, MA: Boyd and Fraser, 1993.
[9] M. R. Garey and D. S. Johnson, *Computer and Intractability: A Guide to the Theory of NP-Completeness*. New York: Freeman, 1979.
[10] I. Ghosh, N. K. Jha, and S. Dey, "A low overhead design for testability and test generation technique for core-based systems," in *Proc. 1997 Int. Test Conf.*, 1997, pp. 50–59.
[11] T. Gonzales and S. Sahni, "Open shop scheduling to minimize finish time," *J. Assoc. Computing Machinery*, vol. 23, pp. 665–679, Oct. 1976.
[12] H. K. Lee and D. S. Ha, "On the generation of test patterns for combinational circuits," Dept. of Electrical Eng., Virginia Polytechnic Institute and State University, Tech. Rep. 12_93, Dec. 1993.
[13] E. J. Marinissen, R. Arendsen, G. Bos, H. Dingemanse, M. Lousberg, and C. Wouters, "A structured and scalable mechanism for test access to embedded reusable cores," in *Proc. Int. Test Conf.*, 1998, pp. 130–143.
[14] H. P. Williams, *Model Building in Mathematical Programming*, 2nd ed. New York: Wiley, 1985.
[15] P. Varma and S. Bhatia, "A structured test re-use methodology core-based system chips," in *Proc. Int. Test Conf.*, 1998, pp. 294–302.
[16] M. Sugihara, H. Date, and H. Yasuura, "A novel test methodology for core-based system LSI's and a testing time minimization problem," in *Proc. Int. Test Conf.*, 1998, pp. 465–472.
[17] Y. Zorian, E. J. Marinissen, and S. Dey, "Testing embedded-core based system chips," in *Proc. Int. Test Conf.*, 1998, pp. 130–143.

Krishnendu Chakrabarty (S'92–M'96) received the B.Tech. degree from the Indian Institute of Technology, Kharagpur, in 1990, and the M.S.E. and Ph.D. degrees from the University of Michigan, Ann Arbor, in 1992 and 1995, respectively, all in computer science and engineering.

He is now Assistant Professor of Electrical and Computer Engineering at Duke University, Durham, NC. His current research projects (supported by NSF, DARPA and industrial sponsors) are in system-on-a-chip test, real-time operating systems, distributed sensor networks, thermal management in integrated circuits, and architectural optimization of microelectrofluidic systems. He has published over 40 papers in archival journals and refereed conference proceedings, and he holds a US patent in built-in self-test.

Dr Chakrabarty is a recipient of the National Science Foundation (NSF) Early Faculty (CAREER) award, and a Mercator Professor award from the Deutsche Forschungsgemeinschaft, Germany, for carrying out research at the University of Potsdam during 2000–2001. He is a member of Sigma Xi, serves as Vice Chair of Technical Activities in IEEE's Test Technology Technical Council, and is a member of the program committees of several IEEE/ACM conferences and workshops.