Test Scheduling in an IEEE P1687 Environment with Resource and Power Constraints

Farrokh Ghani Zadegan[1], Urban Ingelsson[1], Golnaz Asani[1], Gunnar Carlsson[2] and Erik Larsson[1]
[1] Linköping University, Linköping, Sweden
[2] Ericsson AB, Stockholm, Sweden
ghanizadegan@ieee.org, urbin@ida.liu.se, golas045@student.liu.se, erila@ida.liu.se gunnar.carlsson@ericsson.com

***Abstract*—In contrast to IEEE 1149.1, IEEE P1687 allows, through segment insertion bits, flexible scan paths for accessing on-chip instruments, such as test, debug, monitoring, measurement and configuration features. Flexible access to embedded instruments allows test time reduction, which is important at production test. However, the test access scheme should be carefully selected such that resource constraints are not violated and power constraints are met. For IEEE P1687, we detail in this paper session-based and session-less test scheduling, and propose resource and power-aware test scheduling algorithms for the detailed scheduling types. Results using the implementation of our algorithms shows on ITC'02-based benchmarks significant test time reductions when compared to non-optimized test schedules.**

***Keywords*-Test Scheduling, Constraints, IEEE P1687, IJTAG**

I. Introduction

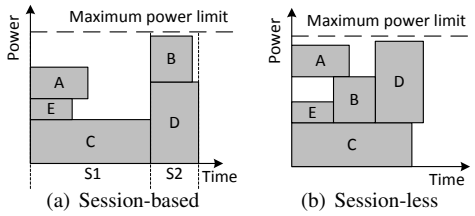
IC manufacturing is advancing and in each new technology generation, ICs are becoming increasingly complex and integrated. A key to successful IC development, at debug, production test, configuration, and in-field test, is access to embedded features, so called instruments, such as phase-locked loops (PLLs), Serializer/Deserializers (SERDESs), temperature sensors, Logic Built-In Self-Tests (LBISTs), Memory Built-In Self-Test (MBIST) controllers, and eFuses (for MBIST repair). A typical PCB of today from Ericsson contains approximately 30 advanced ICs where each IC can contain hundreds of various embedded instruments. In the future, when integration allows the equivalent of such PCBs to be manufactured in a single IC, the total number of instruments for an IC can easily be in the range of several thousands. For low-cost production test and in-field test, the complexity of processing thousands of instruments brings challenges to the problem of test time reduction through test scheduling which is the topic of this paper.

IEEE P1687 (IJTAG) [1] is proposed to enable standardized access to embedded instruments. We envision that P1687 is used in production test to access embedded test features. In contrast to IEEE 1149.1 (JTAG) [2], P1687 provides flexibility to dynamically configure the scan paths through so called segment insertion bits (SIBs). The flexibility from P1687 makes it possible to implement both session-based and session-less schedules, whereas JTAG only allows session-based schedules. A session is a set of tests that start at the same time and the schedule consists of a non-overlapping sequence of such sessions. It is known that for P1687 the fully concurrent schedule leads to the lowest test access time [3] and becomes a cost-saver in IC production test where test cost depends on test application time (TAT). However, fully concurrent scheduling may not be possible due to resource constraints and requirements on power consumption. Hence, there is a need for TAT-optimizing test scheduling that considers resource and power constraints.

This paper analyzes and proposes solutions to the resource- and power-aware test scheduling problem in a P1687 environment. As a prerequisite, identified in the review of prior work (Section II), we develop a test time calculation method for general schedules (Section IV). Both session-based and session-less schedules are considered. Based on our analysis (Section V), we propose three test scheduling algorithms suitable to P1687 (Section VI), categorized by the type of schedules they produce, namely session-based (SB), optimized session-based (OSB) and optimized session-less (OSL). On implementations of these algorithms, experimental results on ITC'02-based benchmarks (Section VII and Section VIII) show significant reduction in TAT with the optimizing algorithms OSB and OSL compared to the non-optimizing SB. This shows that the optimizing algorithms effectively solve the considered test scheduling problem.

II. Prior Work

Significant research has been done on test scheduling [4]–[6]. Chou *et al.* [4] discusses for general VLSI systems a graph-based approach to test scheduling that takes resource constraints and power limits into account. Zorian [5] proposes for JTAG-based systems a session-based scheduling technique for the application of built-in self-tests (BISTs). A session-based schedule is illustrated in Fig. 1(a), where the rectangles represent tests with power dissipation and test time. Sessions are used to group tests that can be applied concurrently within the maximum power limit. In general, sessions are used to separate tests that would otherwise conflict due to resource constraints. For built-in self tests, the duration of a test session is determined by the longest test in the session, and TAT is the sum of the session durations. The technique in [5] groups tests into sessions such that hardware cost and power consumption are controlled. Muresan *et al.* [6] proposed another technique to addresses the same problem as Chou *et al.* [4] and Zorian [5]. Different from the approaches by Chou *et al.* [4] and Zorian [5], Muresan *et al.* [6] proposes a session-less scheduling technique. The concept of a session-less schedule



(a) Session-based (b) Session-less

Fig. 1. Schedule types

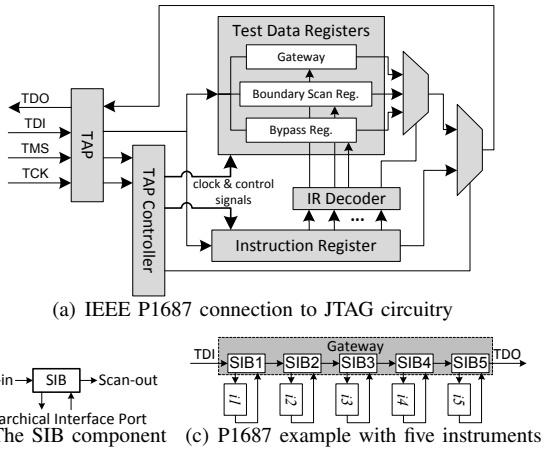
is illustrated in Fig. 1(b). The test access mechanism to implement a session-less schedule is not detailed in [6]. A session-less test schedule is not generally possible with JTAG and a single JTAG test access port (TAP). This is because with a single TAP, concurrent testing can only be implemented by including the tested components in the same JTAG test data register (TDR) and only one TDR can be accessed at a time. In contrast, P1687 allows a session-less test schedule, because P1687 offers flexibility in terms of configuring the scan path, which is not available to JTAG.

In Chou *et al.* [4], Zorian [5], and Muresan *et al.* [6], it is assumed that the duration of a given test is constant. This assumption does not hold in a P1687 environment in which a test duration depends on the other tests that are scheduled concurrently. Therefore, neither the test scheduling methods described in [4]–[6], nor the TAT calculation schemes used there, directly apply to a P1687 environment. The lack of a test time calculation scheme for P1687 was addressed by Zadegan *et al.* [3]. However, only fully sequential schedules or fully concurrent schedules are addressed; and to evaluate a general schedule, a method for calculating TAT for general schedules is needed. Further, no work has addressed test scheduling with resource and power constraints for P1687, which is the topic of this paper. We minimize TAT, given a P1687 network, a set of instruments I, and power and resource constraints. Each instrument $i \in I$ has a number of test patterns tp_i, a scan-chain length l_i, and a peak power dissipation value p_i that must be taken into account whenever instrument i is active.

To perform power-constrained scheduling, a test power approximation model is required as discussed in [6]. In this paper, we consider the peak power value for each instrument. The sum of peak power values for simultaneously active instruments should never exceed the maximum power limit. Compared to the power dissipation of the instruments, we consider the power dissipation of P1687 circuitry to be negligible.

III. Background

P1687 proposes a standard for access to on-chip instruments through the JTAG TAP with an additional instruction called GateWay ENable (GWEN), which activates a test data register called Gateway. Fig. 2(a) shows the JTAG circuitry with the Gateway register. When activated, the Gateway opens up to a P1687 network onto which instruments are connected, such as in the example in Fig. 2(c). To build the network, P1687 details a component called Segment Insertion Bit (SIB), see Fig. 2(b), which acts as a 1-bit register on the scan path during shifting. During the JTAG Update-DR operation, the bit currently inside the SIB sets the state of the SIB. Either the SIB is closed and data is shifted straight through, or the SIB is open and data is shifted to the P1687 network segment that is connected on the



(a) IEEE P1687 connection to JTAG circuitry

(b) The SIB component (c) P1687 example with five instruments

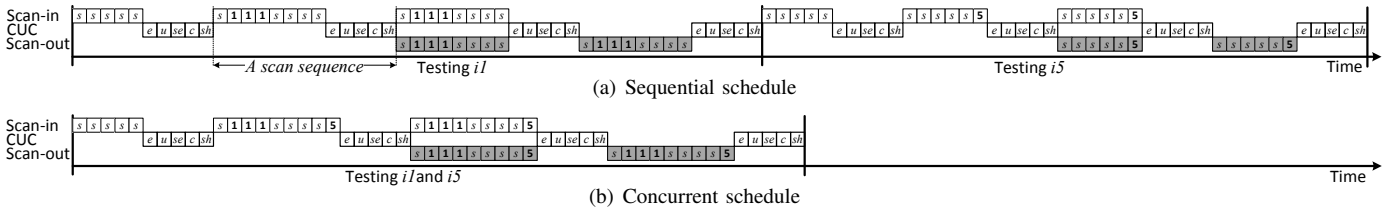
Fig. 2. IEEE P1687 overview

SIB's Hierarchical Interface Port. A P1687 network example is presented in Fig. 2(c). In this example, the SIBs form the Gateway register as can be seen by comparing Fig. 2(c) with Fig. 2(a).

By using the example of Fig. 2(c), it will be described in the following how tests can be applied sequentially and concurrently in a P1687 environment. Fig. 3 shows in detail the required steps. For P1687, TAT consists of time transporting test data (blocks numbered according to the instruments that are given the test data) and two types of overhead [3], namely SIB programming overhead (s blocks) and JTAG protocol overhead (blocks in the middle row). The SIB programming overhead is the time spent transporting SIB control bits. JTAG protocol overhead is the progression of five states in the TAP controller state machine during apply-and-capture. These five states are Exit1-DR, Update-DR, Select-DR-Scan, Capture-DR, and Shift-DR (e,u,se,c, and sh blocks, respectively). We use CUC (Cycle of Update and Capture) as short for JTAG protocol overhead. As can be seen in Fig. 3(a), each CUC marks the end of a *scan sequence*. Each scan sequence involves two operations. Firstly, shifting test data for all active instruments and SIB control bits, and secondly, applying test stimuli and capturing the corresponding responses.

Assume that instrument $i1$ ($l_1 = 3$ and $tp_1 = 2$) and instrument $i5$ ($l_5 = 1$ and $tp_5 = 2$) are to be tested sequentially. Initially all SIBs are closed. To test instrument $i1$, SIB1 should be programmed to be open and the other SIBs to remain closed. This initial SIB programming is represented in Fig. 3(a) by five leftmost s blocks followed by a CUC. After opening SIB1, the first test stimuli vector for $i1$ can be shifted in along with SIB reprogramming data. After a CUC, the captured test responses can be shifted out which are marked by gray boxes in the figure. The shift-out of the test responses can overlap in time with the next test stimuli. After shifting out the responses for the second test stimuli, testing of $i1$ is complete. Testing $i5$ follows the same procedure as testing $i1$.

The concurrent testing of $i1$ and $i5$ is illustrated in Fig. 3(b). In concurrent testing of $i1$ and $i5$, the initial SIB programming is such that both SIB1 and SIB5 are opened. Opening SIB1 and SIB5 includes the scan-chains of $i1$ and $i5$ in the scan path, which allows the test stimuli vectors for both instruments to be shifted in, and the test responses to be shifted out, at the same



(a) Sequential schedule

(b) Concurrent schedule

Fig. 3. Impact of concurrency on test duration with IEEE P1687

time. By comparing Fig. 3(a) and Fig. 3(b), it can be seen that the time from starting a test to its end, depends on the network (the number of SIBs) and on the other tests that are performed concurrently. Furthermore, in concurrent testing the number of scan sequences is reduced resulting in less SIB programming overhead and less CUC overhead, and consequently lower TAT.

As can be seen from the above example, the test schedule affects the overhead of SIB programming and CUC. In other words, for each scan sequence that brings overhead we want to transport as much test data as possible. Therefore, in this paper, to generate a resource- and power-constrained schedule with minimized TAT, the key idea is to schedule tests such that concurrency is maximized and the number of scan sequences is minimized, and thereby minimize SIB programming overhead and CUC. To measure TAT for the generated schedule, we develop a TAT calculation method.

IV. Test Time Calculation for a Given Schedule

The test time for a schedule in a P1687 environment is not given explicitly due to SIB programming overhead and CUC overhead. Therefore, there is a need of a test time calculation method. In this section, we explain how TAT is calculated for general session-based and session-less schedules.

Zadegan *et al.* [3] proposed algorithms for automated calculation of TAT for fully concurrent schedules and for fully sequential schedules. Here, in a concurrent schedule all tests are started as soon as possible considering the P1687 network, and as soon as a test is finished, the instrument employed in that test is excluded from the scan path. This is done to shorten the scan path for the tests that are still being applied. It is possible to employ the TAT calculation algorithm for fully concurrent schedules proposed in [3], to obtain TAT for a general session-based schedule, as is explained in the following. A general session-based schedule can be seen as a succession of sessions, and TAT is the sum of the TATs for each of the sessions. Since in each session tests are performed concurrently, it is possible to use the algorithm for concurrent schedule proposed in [3], to calculate TAT for each of the sessions separately, and obtain the TAT for the whole schedule by summing up these values. To calculate the test time for each individual session, using the algorithm in [3], the required input is the number of test patterns for each instrument. For the session's active instruments, we use the given number of test patterns (tp_i), and for all other (inactive) instruments we use -1 (for reasons detailed in [3]).

To explain how TAT for a general session-less schedule can be calculated, the following example is provided. Consider the five instruments in the network shown in Fig. 2(c). Table I lists the properties for the instruments. Assume that these instruments are to be tested according to a given session-less schedule which is shown in Fig. 8(b). We represent a session-less schedule as a succession of *virtual sessions* and a set of rules for how to practically apply the schedule in a P1687 environment. The succession of virtual sessions of the schedule in Fig. 8(b) is presented in Fig. 4. Each virtual session describes (1) a set of instruments that are tested concurrently and (2) a number of test patterns to apply to that set of instruments. A virtual session is a step of the schedule in which the set of active instruments is constant. It should be noted that if a test for an instrument is started but not completed in one virtual session, it will continue in the next virtual session.

```
Virtual session S1: i1, i5  : 2 test patterns
Virtual session S2: i5, i3  : 1 test pattern
Virtual session S3: i4, i3  : 1 test pattern
Virtual session S4: i4      : 1 test pattern
Virtual session S5: i2      : 1 test pattern
```

Fig. 4. Representation of a session-less schedule, using a succession of virtual sessions

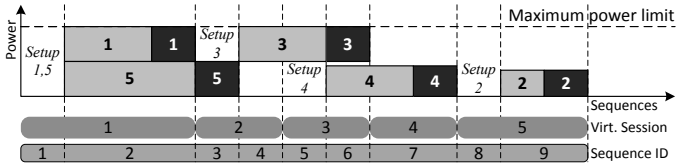


Fig. 5. A P1687-specific schedule derived from the schedule in Fig. 4, based on the network in Fig. 2(c)

The succession of virtual sessions in Fig. 4 abstracted away from the P1687-specific steps required (1) to configure the P1687 network before accessing any of the instruments and (2) to shift out the responses for the last test pattern from an instrument, before closing the SIB for that instrument. These steps, however, should be considered in precise calculation of TAT. Therefore, a P1687 network-specific representation of a given schedule is required before TAT can be calculated. Fig. 5 shows one such P1687-specific schedule based on the network given in Fig. 2(c). The vertical axis shows the power dissipation of the tests and the horizontal axis marks the scan sequences required to implement the schedule. In Fig. 5, the tests are represented with grayed rectangles marked by ID of the instrument each test belongs to, the shift-out of the last test responses for each test is represented with rectangles with inverted colors, and the required network configuration for each instrument is represented with an empty slot marked as "*Setup*". In Fig. 5, the "Virt. Session" denotes the virtual session to which the corresponding scan sequence belongs.

In this work, to obtain one such P1687-specific schedule from a given representation (Fig. 4), the following two rules are applied:

1) If any instrument, from the set of the instruments for a virtual session, has not been activated in the previous

TABLE I
PROPERTIES FOR THE INSTRUMENTS IN FIG. 2(C)

Instrument	$i1$	$i2$	$i3$	$i4$	$i5$
Number of test patterns (tp)	2	1	2	2	3
Scan-chain length (l)	3	7	3	3	1
Peak power dissipation (p)	8	6	8	7	8

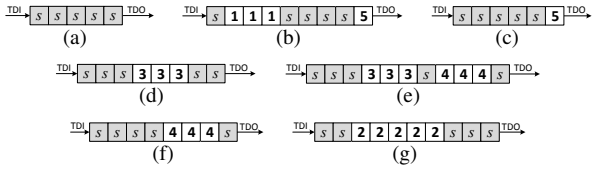


Fig. 6. Scan path configurations for the network shown in Fig. 2(c)

virtual sessions (i.e. its corresponding SIB is still closed), the required configuration scan sequences are added to the schedule. An example is "*Setup 1,5*" before applying tests to $i1$ and $i5$ in Fig. 5.

2) If in the beginning of a virtual session, the remaining number of patterns for an instrument mentioned for that virtual session, is equal to the number of patterns specified for the virtual session, i.e. its test is completed by the end of this virtual session, one sequence is added to the schedule to complete the test for that instrument by performing the last shift-out. An example is the inverted rectangle marked by 1, which is added to represent the last shift-out for $i1$.

Table II will be used to describe the steps, i.e. scan sequences, required to apply the test patterns according to the P1687-specific schedule. Each row of Table II represents one scan sequence or a number of subsequent identical scan sequences, as marked by "# of scan sequences". "Seq. ID" in Table II indicates the order of the required scan sequences. "Scan-path" refers to the part of Fig. 6 which shows the scan-path configuration corresponding to the scan sequence. "Virtual Session" enumerates virtual sessions. "SIBs" presents the number of SIBs on the scan-path for each scan sequence. "Instruments" presents the number of bits scanned for the active instruments in the scan sequence. "$\sum$" shows the total number of bits scanned per sequence. "CUC" shows the number of test clock cycles (TCKs) spent on performing an apply-and-capture for each scan sequence. Finally, "Sum for scan-path" presents the total number of clock cycles that are required for each scan-path configuration. The last row of Table II, presents TAT which is the sum of the values in the last column.

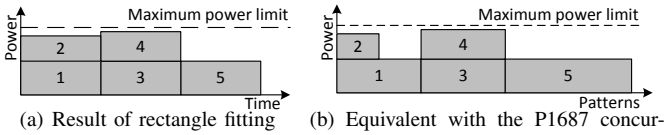
From the above, it can be seen that a representation of a session-less schedule and two rules can be used to derive a P1687-specific schedule. From such a P1687-specific schedule, TAT is calculated by automating the process detailed in Table II. In the remainder of this paper, we use this TAT calculation method to analyze and evaluate test scheduling approaches.

V. SCHEDULING ANALYSIS

To see how existing test scheduling approaches perform in a P1687 environment, consider the following. The typical approach in [4]–[6] is to view the tests as rectangles described by the test's power dissipation (rectangle's height) and the test's duration in time units (rectangle's width). The test

TABLE II
TEST TIME CALCULATION STEPS FOR THE SCHEDULE GIVEN IN FIG. 4

Seq. ID	**Scan-path**	**Virtual Session**	**Scanned bits SIBs**	**Scanned bits Instruments**	**$\sum$**	**CUC**	**# of scan sequences**	**Sum for scan-path**
1	Fig. 6(a)	S1	5	0	5	5	1	$(5+5)\cdot 1$
2	Fig. 6(b)	S1	5	4 (l_1+l_5)	9	5	3	$(9+5)\cdot 3$
3	Fig. 6(c)	S2	5	1 (l_5)	6	5	1	$(6+5)\cdot 1$
4	Fig. 6(d)	S2	5	3 (l_3)	8	5	1	$(8+5)\cdot 1$
5	Fig. 6(d)	S3	5	3 (l_3)	8	5	1	$(8+5)\cdot 1$
6	Fig. 6(e)	S3	5	6 (l_3+l_4)	11	5	1	$(11+5)\cdot 1$
7	Fig. 6(f)	S4	5	3 (l_4)	8	5	2	$(8+5)\cdot 2$
8	Fig. 6(a)	S5	5	0	5	5	1	$(5+5)\cdot 1$
9	Fig. 6(g)	S5	5	7 (l_2)	12	5	2	$(12+5)\cdot 2$
Test application time (TAT)								$\sum$=175



(a) Result of rectangle fitting (b) Equivalent with the P1687 concurrency concept

Fig. 7. Example analyzing rectangle packing based approaches

scheduling problem is to fit the rectangles in a strip limited on one side by the maximum power limit, and minimize TAT. Such an approach, when employed in a P1687 environment, could result in an unnecessarily long schedule as is shown in the following example. Consider the five instruments in Fig. 2(c) which are described in Table I. By considering each individual instrument i, the test duration τ_i can be calculated as:

$$\tau_i = tp_i \cdot (l_i + CUC) + l_i \tag{1}$$

In this example, all five instruments have a test duration of 19 time units. Therefore, given the maximum power limit of 16 units, the schedule shown in Fig. 7(a) is a good schedule according to [4]–[6]. We would interpret Fig. 7(a) as follows. Test 1 and test 2 are performed concurrently, succeeded by concurrent application of tests 3 and 4, succeeded by test 5. The schedule in Fig. 7(a) is also shown in Fig. 7(b) but here the horizontal axis shows the number of test patterns. TAT is 175 time units calculated as in Section IV. A better schedule can be seen in Fig. 8(c) which results in TAT of 155 time units. This example shows that the approaches from [4]–[6] lead to suboptimal schedules when applied in a P1687 environment. It can be explained by considering that performing tests concurrently in a P1687 environment impacts the duration of individual tests, see Section III. Consequently, the tests cannot be viewed as rectangles where the width is specified by test duration.

To find a better view of tests, the following should be considered. In Section III, it is noted that performing tests concurrently reduces TAT, compared to when the same tests are performed sequentially. The benefit of concurrency in P1687 is not depending on how long time tests are running together, but rather on the fact that test patterns are applied together sharing the same SIB programming and CUC overhead. In the following we view tests as rectangles with widths specified in number of test patterns (tp). Using this view of tests, the problem of power-constrained test scheduling with the objective of minimizing TAT, can be described as the classic strip packing problem which is NP-hard. Since in general, we do not make any assumptions about resource constraints, the problem complexity remains the same, when considering both

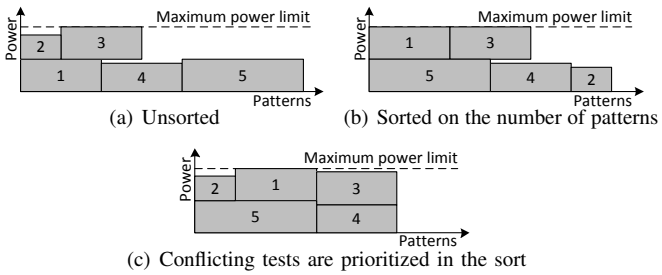
power and resource constraints. To solve the problem with both power and resource constraints, in the following, heuristics will be proposed.

The basic session-based test scheduling approach for resolving power and resource constraints starts with an empty schedule. From a given list, tests are moved into the schedule at *start time* zero such that only those tests that can be run concurrently are moved, considering power and resource constraints. Subsequently, the end of the scheduled test with the most patterns is considered the new *start time*, and the process is repeated for the remaining list of tests. The process continues until the list of tests is empty. It should be noted that in a P1687 environment, two approaches might be assumed for applying tests according to a session-based schedule. In the first approach, no change in the P1687 network configuration is made within a session, and therefore, instruments employed in the test remain on the scan-path until the end of session. In the second approach, instruments are excluded from the scan-path as soon as their testing is finished.

To get a session-less test schedule, a similar approach can be employed. The difference is that when updating the *start time*, it is set to the earliest time when any of the tests finishes. Fig. 8(a) shows an example of a session-less schedule generated by this approach. The example is based on the instruments in the P1687 network shown in Fig. 2(c), detailed in Table I, with a power limit of 16 units and resource conflicts between tests 4 and 5, and between tests 2 and 4. The TAT for this schedule is 185 time units. In Fig. 8(a) the scheduling process starts by adding test 1 to the empty schedule. After adding also test 2, no more tests can fit within the maximum power limit. Therefore, test 3 is scheduled after test 2 which was the earliest test to finish. Similarly, test 4 is scheduled after test 1, but test 5 cannot be scheduled after test 3 because that would cause a resource conflict with test 4.

To avoid the situation in Fig. 8(a) where the longest test is performed last without any concurrency, we sort the tests in a descending order based on the number of patterns. Based on this sorting we apply the same approach as above. The result is shown in Fig. 8(b). Here, TAT is 175 time units. From Fig. 8(b) it can be seen that considering the conflicting tests (here 2 and 4) late in the scheduling process limits the possibility for concurrency. Therefore, the next example (Fig. 8(c)) analyzes the impact of considering the conflicting tests earlier than the non-conflicting tests in the scheduling. We consider the test list 5 4 2 1 3. Here, test 5 is added first because it has the most test patterns and it has a conflict with test 4. The next test is not test 4 because of the resource conflict with test 5. Instead, test 2 is added to the schedule. Similarly, test 1 follows after test 2, because test 4 can still not be applied. Finally tests 4 and 3 are added. The schedule in Fig. 8(c) results in TAT of 155 time units.

In this section we have seen that the important parameters are power dissipation and the number of test patterns for each test. Furthermore, we analyzed an approach for test scheduling that resolves power- and resource-constraints, and observed in the examples of Fig. 8 that TAT, can be reduced by prioritizing tests with conflicts and tests with many patterns. It is interesting to see from Fig. 8 that the length of the schedules in number of patterns is correlated to the length of schedules in time units.



(a) Unsorted (b) Sorted on the number of patterns

(c) Conflicting tests are prioritized in the sort

Fig. 8. Example analyzing the effect of sorting

Algorithm 1: Optimized session-based (OSB) scheduling

Input: $Instruments$ as $\{(i1, tp_{i1}, p_{i1}, rc_{i1}), \ldots\}$
Input: $ResourceConflicts$ as $\{(im, in), (im, io), \ldots\}$
Input: $PowerLimit$

Output: $Sessions$ as $\{(tp_{S1}, \{in, io, \ldots\}), (tp_{S2}, \{im, \ldots\}), \ldots\}$

```
1  Sort Instruments on rc then on tp, both in descending order;
2  Sessions := {};
3  while Size(Instruments) > 0 do
4      s := {} ;                                  // Current session
5      p_s := 0 ;                        // Power consumption for s
6      tp_s := 0 ;                  // Number of test patterns for s
7      foreach (i, tp_i, p_i, rc_i) ∈ Instruments do
8          if i has no constraints with any instrument in s then
9              if p_s + p_i ≤ PowerLimit then
10                 s := s ∪ {i};
11                 tp_s := max(tp_s, tp_i);
12                 p_s := p_s + p_i ;
13                 Remove (i, tp_i, p_i, rc_i) from Instruments;
14             end
15         end
16     end
17     Sessions := Sessions ∪ {(tp_s, s)}
18 end
```

VI. Method

In this section, based on the analysis presented in Section V, three scheduling algorithms will be presented, namely session-based (SB), optimized session-based (OSB), and optimized session-less (OSL). The SB and OSB algorithms are very similar and therefore, will be described together in Section VI-A. The OSL algorithm will be described in Section VI-B.

A. Session-based and Optimized Session-based Scheduling

The SB and OSB algorithms are very similar, with the difference being an initial ordering of instruments in case of the OSB algorithm. Therefore, only the OSB algorithm is presented as Algorithm 1, and as for the SB algorithm only the differences are highlighted.

Algorithm 1 describes the steps to generate an optimized session-based (OSB) schedule, given (1) a list of instruments to be tested, (2) a list of resource conflicts, and (3) a power constraint. The list of instruments is a set, where each element is an instrument specified by a tuple as (i, tp, p, rc). In tuple (i, tp, p, rc), i is a unique instrument ID, tp is the number of patterns for the instrument, p is the peak power dissipation when the instrument is active, and rc will be 1 if the instrument has any resource conflicts with any other instrument, and 0 otherwise. The resource constraints are given as a set of (im, in) tuples where im and in are the IDs for two instruments that have a resource conflict.

Algorithm 2: Optimized session-less (OSL) scheduling

Input: $Instruments$ as $\{(i1, tp_{i1}, p_{i1}, rc_{i1}), \ldots\}$
Input: $ResourceConflicts$ as $\{(im, in), (im, io), \ldots\}$
Input: $PowerLimit$

Output: $VirtualSessions$ as
$\{(tp_{s1}, \{in, io, \ldots\}), (tp_{s2}, \{im, \ldots\}), \ldots\}$

```
1  Sort Instruments on rc then on tp, both in descending order;
2  VirtualSessions := {};
3  while Size(Instruments) > 0 do
4      vs := {} ;                    // Current virtual session
5      p_vs := 0 ;                   // Power consumption for vs
6      tp_vs := ∞ ;                  // Number of patterns for vs
7      foreach (i, tp_i, p_i, rc_i) ∈ Instruments do
8          if i has no constraints with any instrument in s then
9              if p_vs + p_i ≤ PowerLimit then
10                 vs := vs ∪ {i};
11                 p_vs := p_vs + p_i;
12                 tp_vs := min(tp_vs, tp_i);
13             end
14         end
15     end
16     VirtualSessions := VirtualSessions ∪ {(tp_vs, vs)};
17     foreach (i, tp_i, p_i, rc_i) ∈ Instruments where i ∈ vs do
18         tp_i := tp_i - tp_vs;
19     end
20     Remove all elements having tp = 0 from Instruments;
21 end
```

The output of the algorithm is a set of sessions where each session is specified as a tuple in the form of $(tp, \{in, io, \ldots\})$. In this tuple, tp specifies the number of test patterns that are applied in the session and $\{in, io, \ldots\}$ specifies the instruments that are active in the session.

The algorithm starts by sorting in descending order the list of instruments, i.e. $Instruments$, on rc then on tp (Line 1). This initial ordering is based on the analysis in Section V and is the key to the optimized TAT, i.e. it is only performed in case of the OSB—and not the SB—scheduling. Sorting on rc separates those instruments having any resource constraints from those having no constraint, and prioritizes the former group over the latter in the scheduling. Inside each group, sorting on tp prioritizes those instruments with larger number of test patterns over those having fewer test patterns.

In Algorithm 1, in each iteration of the main loop (Lines 3-18), a new session s is created (Line 4) and instruments are assigned to it (Line 10). Any instrument added to a session is removed from the $Instruments$ set (Line 13). The newly created session s is finally added to the $Sessions$ set (Line 17) before starting a new iteration. Before assigning an instrument to a session, power and resource constraints should be checked (Lines 8-9). The total power dissipation of a session is calculated and stored in the p_s variable (Line 12). The maximum number of test patterns found among the instruments in session s is recorded in tp_s (Line 11).

B. Optimized Session-less Scheduling

Compared to session-based scheduling where all tests in a session are started at the same time, in session-less scheduling tests can start independent from each other. Algorithm 2 is similar to Algorithm 1, except for the fact that instead of sessions there are virtual sessions that are introduced to represent a session-less schedule (see Section IV). The corresponding differences are in Lines 12, 18, and 20 of Algorithm 2.

The main difference is that—unlike Algorithm 1—once an instrument is assigned to a virtual session, it is not removed from $Instruments$. Instead, depending on the number of test patterns of the current virtual session (tp_{vs}), the number of test patterns for that instrument is modified (Line 18) and the rest of the test patterns, if any, will be kept for the next virtual session. If, however, all the (remaining) test patterns for an instrument are assigned to the current virtual session, that instrument will be removed from $Instruments$ (Line 20). Another difference is that in determining the number of test patterns of the current virtual session, the lowest number of test patterns for the instruments inside that partition is used (Line 12) which is in contrast to Algorithm 1 where the largest number of test patterns for the instruments assigned to a session determines the length of that session, i.e. tp_s.

VII. Experimental Setup

We have performed experiments to evaluate the capability of the proposed SB, OSB, and OSL algorithms in reducing TAT. To perform the experiments, we have selected instruments based on the ITC'02 benchmark set [7]. Each SOC from the ITC'02 set contains a number of cores, where each core has a number of I/O pins as well as some internal scan-chains. For our experiments, we regarded the set of I/O pins for each core, and each of the scan-chains inside that core, as instruments. As discussed above, test scheduling for P1687 is significantly different from test scheduling for core-based SOCs.

From the ITC'02 benchmarks, the d695, p22810, p34392, and p93791 SOCs are considered and the sets of instruments extracted from these SOCs are called A, B, C, and D, respectively. In Section VIII experimental results are presented for these sets. In extracting the above-mentioned sets, the power consumption for each of the cores inside d695, p22810 and p93791 are taken from [8], and it is assumed that all the instruments inside each core consume the same amount of power. As for instruments inside p34392, the power consumption for each instrument is assumed as a number proportional to the length of the shift-register for that instrument.

The algorithms proposed in Section VI are implemented and employed in experiments on A, B, C and D sets, thereby generating SB, OSB and OSL schedules for each of these sets. Four maximum power limits and two sets of resource constraints are considered. The number of conflicts in each set of constraints is reported in Table III in parentheses for each of the "Resource constraint sets".

The TAT for each of the generated schedules is calculated and reported by assuming a P1687 network similar to the one in Fig. 2(c). The results of the experiments are presented and discussed in Section VIII.

VIII. Experimental Results

Table III lists the generated instrument sets (leftmost column) and the number of instruments in each set. "Algorithm" lists the generated schedules, correspondingly named after the algorithm used to generate them, for each of the considered instrument sets. "Test application time" presents the TAT calculated for the corresponding schedule, under different power constraints (Column "PC"), and under either no

TABLE III
EXPERIMENTAL RESULTS

Set of Instruments†	Algorithm	Test application time (in million TCKs)											
		No resource constraint				Resource constraints set 1 ($\sim 50^*$)				Resource constraints set 2 ($\sim 190^*$)			
		PC=∞	PC=1000	PC=850	PC=680	PC=∞	PC=1000	PC=850	PC=680	PC=∞	PC=1000	PC=850	PC=680
A (147) based on d695	SB ‡	1.97	1.15	1.36	1.37	2.01	1.31	1.38	1.38	2.03	1.30	1.50	1.44
	SB	0.74	0.83	0.86	0.90	0.78	0.86	0.87	0.91	0.82	0.85	0.88	0.92
	OSB	0.74	0.81	0.82	0.84	0.77	0.84	0.84	0.89	0.83	0.88	0.89	0.92
	OSL	0.74	0.80	0.81	0.84	0.76	0.81	0.82	0.85	0.81	0.82	0.82	0.84

		No resource constraint				Resource constraints set 1 ($\sim 30^*$)				Resource constraints set 2 ($\sim 50^*$)			
		PC=∞	PC=650	PC=450	PC=250	PC=∞	PC=650	PC=450	PC=250	PC=∞	PC=650	PC=450	PC=250
B (224) based on P22810	SB ‡	383	28	29	24	343	28	29	24	308	28	29	24
	SB	11	15	15	17	11	15	15	17	11	15	15	17
	OSB	11	13	13	16	11	14	13	17	11	14	14	17
	OSL	11	11	11	14	11	11	12	14	11	12	12	15

		No resource constraint				Resource constraints set 1 ($\sim 40^*$)				Resource constraints set 2 ($\sim 80^*$)			
		PC=∞	PC=1500	PC=1150	PC=850	PC=∞	PC=1500	PC=1150	PC=850	PC=∞	PC=1500	PC=1150	PC=850
C (82) based on P34392	SB ‡	290	53	55	51	275	51	46	51	248	74	53	58
	SB	18	20	21	22	19	20	20	22	19	21	21	23
	OSB	18	19	19	19	18	20	22	21	18	20	20	20
	OSL	18	18	18	19	18	18	18	19	18	18	18	19

		No resource constraint				Resource constraints set 1 ($\sim 50^*$)				Resource constraints set 2 ($\sim 380^*$)			
		PC=∞	PC=1500	PC=1000	PC=550	PC=∞	PC=1500	PC=1000	PC=550	PC=∞	PC=1500	PC=1000	PC=550
D (554) based on P93971	SB ‡	622	113	71	76	591	89	71	76	618	133	91	91
	SB	35	47	50	62	37	45	50	62	48	59	62	74
	OSB	35	41	44	54	35	43	48	58	48	49	52	60
	OSL	35	39	43	54	35	39	43	54	45	45	45	54

*Size of the resource constraint set, i.e. number of resource conflicts.
†The numbers inside parentheses, denote the number of instruments in the corresponding set.
‡TAT is calculated assuming that network configuration is not changed within a session.

resource constraint or one of the sets of constraints described in Section VII.

For the SB schedule, TAT is reported for both when the network configuration is not changed within a session ("SB‡"), and when network is reconfigured within a session ("SB") to exclude instruments from the scan-path as soon as their test is finished. Comparing "SB‡" and "SB", reveals that in all cases, "SB‡" shows a higher TAT than "SB". The reason is that for SB‡ the P1687 network configuration is not changed within a session, which requires scanning in dummy bits for instruments whose tests are finished earlier than the other tests in the same session. Therefore, it can be seen that employing the flexible P1687 scan path helps achieve lower TAT.

For PC=∞ and no resource constraints, the generated schedules are fully concurrent independent of the algorithm. Consequently, SB, OSB and OSL have the same TAT. In this case, an observation regarding "SB‡" is that all the instruments remain on the scan path while the instrument having the largest number of patterns is being tested—thus requiring a large number of dummy bits.

By using OSB rather than SB, TAT can be reduced by up to 18% for instrument set **D**, resource constraint set 2, and PC=550, and reduced on average by 5%. However, we have seen three cases, i.e. instrument set **A**, resource constraint set 2, and for PC=∞, PC=1000, and PC=850, for which SB performs better than OSB.

The general observations regarding TAT for the SB, OSB, and OSL schedules are that (1) reduced power constraint ("PC") increases TAT since it limits the number of tests that can be performed concurrently, and (2) in all of the cases, OSL results in the lowest TAT. The best result with OSL is 27% reduction in TAT compared to SB, for instrument set **D**, resource constraint set 2, and PC=550.

IX. CONCLUSION

We envision that in the future, production test will involve accessing thousands of embedded test features using the upcoming standard IEEE P1687, which unlike IEEE 1149.1 provides the flexibility to implement both session-based and session-less schedules. In this context we have addressed the test scheduling problem with resource and power constraints to minimize TAT. Based on thorough analysis of the impact of P1687 on test scheduling, our contributions are (1) a TAT calculation method for general schedules in a P1687 environment, and (2) development and implementation of three power- and resource-aware test scheduling algorithms, i.e. session-based (SB), optimized session-based (OSB), and optimized session-less (OSL). With SB as a baseline, experimental results demonstrate the capability of OSB and OSL to reduce TAT by up to 18% and 27%, respectively. Furthermore, OSL always performed better than OSB in terms of TAT reduction.

REFERENCES

[1] IJTAG, "IJTAG - IEEE P1687," Mar. 2010. [Online]. Available: http://grouper.ieee.org/groups/1687

[2] IEEE association, "IEEE Std 1149.1-2001, IEEE Standard Test Access Port and Boundary-Scan Architecture," 2001.

[3] F. G. Zadegan, U. Ingelsson, G. Carlsson, and E. Larsson, "Test Time Analysis for IEEE P1687," in *Proceedings of the IEEE Asian Test Symposium (ATS)*, 2010, pp. 455–460.

[4] R. M. Chou, K. K. Saluja, and V. D. Agrawal, "Scheduling tests for VLSI systems under power constraints," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 5, no. 2, pp. 175–185, Jun. 1997.

[5] Y. Zorian, "A Distributed BIST Control Scheme for Complex VLSI Devices," in *Proceedings IEEE VLSI Test Symposium (VTS)*, Princeton, NJ, USA, Apr. 1993, pp. 6–11.

[6] V. Mureşan, X. Wang, V. Mureşan, and M. Vlăduţiu, "Greedy Tree Growing Heuristics on Block-Test Scheduling Under Power Constraints," *J. Electron. Test.*, vol. 20, pp. 61–78, February 2004.

[7] E. J. Marinissen, V. Iyengar, and K. Chakrabarty, "A set of benchmarks for modular testing of SOCs," in *Proceedings of the International Test Conference*, 2002, pp. 519–528.

[8] J. Pouget, E. Larsson, and Z. Peng, "Multiple-constraint driven system-on-chip test time optimization," *J. Electron. Test.*, vol. 21, pp. 599–611, December 2005.