Power-constrained Test Scheduling for SoCs under a "no session" scheme

Marie-Lise Flottes, Julien Pouget, Bruno Rouzeyre
Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier, 161 rue Ada, 34392 Montpellier cedex 5, France

Abstract: This paper considers the scheduling problem of core tests in a system. Our objective is to minimize the total system test time while respecting system constraints in terms of power consumption and test resource sharing. A simple and effective scheduling heuristic is proposed based on a no sessions based scheme for better overall test time optimisation.

Key words: System-on-Chip, test scheduling

1. INTRODUCTION

Present need of high-performance complex systems leads to modify the classical design style, based on standard IC assembling, to System-on-a-chip design (SoC), based on core and user-define logic (UDL) integration. While individual functions were tested before assembling in the classical approach, the SoC design technique counter part is that it postpones the test of every core and UDL at the end of the system implementation leading thus to prohibitive test time. In this context, an optimised test scheduling of individual SoC functions allows minimizing the system test time and thus the system cost. Obviously, the minimal system test time would be achieved by simultaneously testing all the individual functions (cores for simplicity). But, most of the time, design constraints prevent this full parallelism. Individual tests are conflicting because 1/ they share common test resources (e.g. test bus, test response compactors…), or 2/ the power consumption during simultaneous testing exceeds the device power allowance.

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: 10.1007/978-0-387-35597-9_40
M. Robert et al. (eds.), *SOC Design Methodologies*

Since many years, numerous techniques have been reported in the literature for testing individual functions and thus cores (memories, processors, application specific functions...). An overview of BIST methods for several core types can be found in [1] for instance. More recently, mechanisms for accessing SoC cores have also been explored [2], [3], [4], [5]. Test access mechanism provides access to the cores from the test "sources" and "sinks" (system IOs in external testing, test pattern generators and response compactors in BIST) but limits the parallelism of test application.

The power consumption is another factor that may impact the test parallelism. Test power dissipation is a function of time, it depends of the switching activity resulting from the application of a new test vector to the system. For simplicity, the core power dissipation can be assigned to a fixed value [6]. A pessimistic point of view consists in defining this value as the maximum power dissipation over all test vectors send to the core (peak power). With this assumption, two cores cannot be tested in parallel if the sum of their peak power goes beyond the maximum allowable system power dissipation, even if their peak power do not occur at the same time! However, such assumption prevents any test parallelism that could damage the system.

This paper presents a heuristic for solving core test scheduling in a system, taking into account the two test parallelism constraints mentioned above. We target the minimal system test time. With this heuristic, test plan is not scheduled into test sessions; every core test starts as soon as resource and power constraints allow it.

The organization of the paper is as follow. In section 2, we present the advantages and drawbacks of several test control schemes: organizing tests in sessions of equal or unequal length, or with no session. Corresponding algorithmic complexities are discussed in section 3. In section 4, we present an algorithm for solving the "no session" scheduling problem. Experimental results are detailed in section 5 and section 6 concludes the paper.

2. TEST CONTROL SCHEMES

The basic approach in test scheduling is to organize tests for the target modules into so-called test sessions. A test session brings together the tests of compatible modules. This compatibility is checked with respect to the test resource sharing needs and the power threshold to not exceed. Related test scheduling techniques assume either equal length test sessions or unequal length test sessions.

With the first assumption, the tests to perform are first assigned to test sessions. After completion, the length of each session can be set to the length of the longest test in that session for further test time optimisation.

This technique does not necessarily lead to the optimal scheduling solution in terms of system test time. Actually, when all test sessions are assumed to have the same length, the scheduling problem consists in minimizing the number of test sessions and not really the overall test time: let's consider an hypothetic SoC composed of three cores, 1, 2 and 3 with respective test lengths Length(1)=1000, Length(2)=800 and Length(3)=500 - test lengths are expressed in terms of number of clock cycles-. Their respective power consumption is 500 mw, the maximal power consumption for the system is 1w and there is no test resource conflict between the cores. In this example, $Max_i Length(i)=1000$ for $i \in \{1,2,3\}$. Consequently, the "equal session length" assumption leads to consider test sessions of 1000 clock cycles. A first scheduling solution Sol1 consists in testing cores C1 and C3 in parallel then to test core C2 in a second test session. A second solution Sol2 consists in testing cores C1 and C2 in parallel then to test C3 in a second test session. There is no solution with only one test session due to the power consumption constraint. The solutions Sol1 and Sol2 are equivalent in terms of number of test sessions for a total test time of 2x1000=2000 clock cycles. Any algorithm based on the assumption of equal length test sessions can equally generate them. However, the test can be stopped after 1000+800=1800 clock cycles in the first solution, and after 1000+500=1500 clock cycles in the second solution.

This example shows that assuming equal lengths for the test sessions and thus targeting the minimal number of test sessions is not sufficient for guarantying the minimal test time at system level.

The second classical scheduling approach assumes unequal length test sessions. In this context, a session length is the time taken to test the core requiring the longest time in the session ($Length(Sj)=Max_{i \in Sj} Length(i)$). With this approach, solution 2 is obtained. Compared to the previous scheme, an additional problem here is to map core tests to test sessions.

Finally, test for cores can be organized without session using a test scheme where the test of every core starts as soon as possible considering resource and power constraints.

Let's compare these two last approaches with help of the following example: the SoC under test includes fourteen cores whose characteristics are given in table 1. Resource constraints are very relaxed in order to let the scheduling solution space sufficiently large (the degree of incompatibility is about 10%). Following pairs (i, j) means that the tests of cores i and j are incompatible due to resource conflicts: (1,3), (1,6), (1,14), (2,9), (2,12), (2,13), (3,9), (3,12), (4,7), (7,12). The power limit is assumed to be Pmax=30 units.

Cores	Test length Di (# clock cycles)	Power consumption during test mode Pi (unit)
1	20000	11
2	11000	9
3	10000	11
4	19000	6
5	10000	16
6	4000	15
7	16000	10
8	7000	2
9	16000	9
10	6000	6
11	9000	7
12	3000	3
13	7000	15
14	5000	8

Table 1. A SoC example: core's test length and power characteristics

2.1 Unequal length test sessions scheduling

According to this session-based scheme, the SoC test example can be scheduled as described in Figure 1. The core 1 fixes the length of the first test session (S1) to 20000 clock cycles. In the same way, the core 9 fixes the length of the second test session, and cores 4, 14 and 5 respectively fix the length of test sessions S3, S4 and S5. The total test length is equal to 72000 clock cycles. For information, this solution represents a gain of 30000 clock cycles compared to a solution generated under the assumption of equal length test sessions.

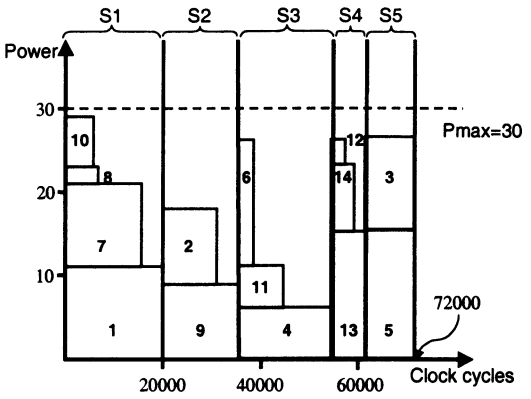

Figure 1. Power and timing report for unequal length session scheduling

2.2 "Sessionless" scheduling

In this scheme, no test sessions are considered. One scheduling solution for the given SoC example is reported in Figure 2. For instance, cores 1, 7 and 9 are simultaneously tested at the beginning of the test mode. The test of core 2 (resp. 8) starts right after the end of the test of core 7 (resp. 9) while test of core 1 is still running. The core-test starts are scheduled as soon as possible considering resource and power conflicts. The system test time is partitioned in several time slots delimited by dashed lines in Figure 2. The maximum allowable system power dissipation is respected for every time slot during the test mode.

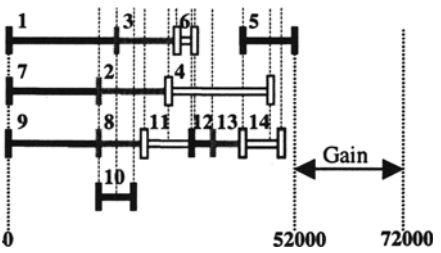

Figure 2. Timing report for the "sessionless" scheduling

Test length is now of 52000 clock cycles, representing an improvement of 20000 clock cycles over the solution based on unequal test session lengths. However, the sessionless scheme is more complex from the synchronization point of view since each core test depends of other(s) core(s) test end signal.

3. SCHEDULING

For all three BIST control schemes, the scheduling problem consists to minimize the total test time while obeying the power constraint limit and the resources sharing possibilities that is, organizing the tests in such a way that:
-1- At any time, the sum of the individual power consumption does not exceed the limit Pmax (constraint $C1$)
-2- the tests of two cores sharing test resources can not overlap in time (constraint $C2$). Constraints $C2$ can be modelled as an incompatibility graph in which nodes represent tests, and an edge links two nodes if they share a test resource.

Let's denote i ={1,2,...,n} the cores to test, Di their test durations, Pi their power consumption, Ti their test starting date and Pmax the maximum system power limit.

3.1 Equal length sessions

Under a equal length session scheme, the scheduling problem can be stated as follows:
Problem P1: Minimize the total test time T_{total} with T_{total} = # sessions*(D_{max}) under constraints $C1$ and $C2$. D_{max} is the duration of the longest core test over all core tests.

As previously mentioned, this problem consists simply to minimize the number of sessions and assign the core tests to the sessions [6] while respecting power and test resource sharing constraints.
Theorem 1: P1 is NP-complete.
Proof: Let's restrict P1 to the case in which $C2$ is null (no resource conflict). The problem sums up to the well-known minimum bin-packing problem (with "bins" being the test sessions). The bin packing problem is NP-complete ([7]). q.e.d.
Remark: by restricting P1 to the case in which Pmax=ΣPi with Pi the individual test power consumptions, i.e. $C1$ is null, P1 sums up to the minimal graph coloring problem which is well-known to be NP-complete. Thus, P1 is made of two interleaved NP-complete problems (bin-packing and minimal graph-coloring).

Heuristics such as the one proposed in [6] can be used for solving P1.

3.2 Unequal length sessions

Whereas in the previous scheme, the assignment of core tests to sessions does not impact on the total test time, this point is crucial here since the

sessions lengths directly depends on it. This point adds an extra difficulty to the problem sated as follows:

Problem P2: Minimize the total test time T_{total} with $T_{total} = \Sigma D(s)$, $s \in$ Sessions, where *D(s)* is given by the longest core test length of the current session s i.e. D(s)=MaxDi, $i \in s$, under constraints $C1$ and $C2$.

Theorem 2: Problem P2 is NP-complete

Proof: Let's consider the case in which all Di are constant. P2 sums up to P1. Thus P2 is NP-complete.

Several scheduling heuristics proposed in the literature (e.g. [8], [9], [10]) can be used for unequal length test session scheduling.

3.3 No sessions

Now the test scheduling problem is stated as follows:

Problem P3: Minimize the total test time T_{total} based on the actual individual test lengths under constraints $C1$ and $C2$.

Theorem 3: P3 is NP-complete.

Proof: Let's restrict P3 to the case in which all Di are constant. P3 sums up to P1. Thus P3 is NP-complete.

In this test scheme, the test scheduling is more complex than in the previous schemes, in which it sums up to a mapping problem of core tests to sessions. Recently, an ILP formulation has been proposed [11] for problem P3 but to the best of our knowledge, no scheduling heuristic have been proposed in the literature for solving it.

It is clear from the example in section 2, that in general a "good" solution for P3 cannot be directly derived from a good solution for P2. In the next section, we propose simple and efficient heuristics for such a control scheme.

4. SCHEDULING HEURISTIC WITH NO SESSIONS

Let's recall that P3 contains P1 that in turn contains the minimal graph-coloring problem.

It is known from the literature that the graph-coloring problem cannot to be approximated in bounded limits when the graph has no special structure [7]. Thus, we developed the following heuristic.

4.1 Algorithm

```
L1 = list of cores sorted by decreasing Di values
L2 = Ø
Tmax=0
While L1 ≠ Ø
  Tmax =Di+Place (first core in L1)
  For all others cores i in L1
    Ti=Place(i)
    if Ti+Di > Tmax
      remove i from the placed cores
      L2 = L2 ∪ {i}
  L1 = L2
```

Figure 3. Test scheduling heuristic

in which function Place (i) positions i as soon as possible i.e. find the earliest date Ti to start i test taking into account that for any other already placed core j such that: (Tj≤Ti and Tj+Dj>Ti) or (Ti≤Tj and Ti+Di>Tj) – i and j tests overlap-:

- ΣPj+Pi<Pmax (*C*1)
- i does not conflict with j (*C*2)

Property 2: the complexity is $O(n^3)$.

Proof: In the worst case, only one core is actually placed in each iteration of the while loop. At the i^{th} iteration, the for-loop iterates i times and executes the Place function which is O(i). Thus, it comes that the complexity is $O(\Sigma(i(n-i))) = O(n^3)$.

5. RESULTS

In order to show the effectiveness of the no session test scheme, we compare our approach with the one presented in [10]. The characteristics of the SoC example used for this comparison are summarised in Table 2. In [10], the authors propose an improvement of the unequal length session approach by allowing several cores to be sequentially tested within a session.

On this example, the system power limit is set to 12 units. It can be seen on Figure 4 that the system test time reduces from 31k clock cycles with the approach presented in [10] to 23k clock cycle with the approach proposed here. In fact, the session-based test scheme results in partitioning the system test time into 4 sessions (S1 to S4), leaving unused time slots that could be used to start test of cores 4 and 3.

Core	Pi	Di	Share test resource with
1	6	16000	6 7 8
2	5	10000	4 5 6 7
3	4	9000	6 7 9
4	2	7000	5
5	8	4000	2 4 7
6	2	3000	1 2 3
7	2	2000	1 2 3 5
8	2	1000	1
9	1	3000	3

Table 2. A SoC example for comparison with a session-based test scheduling approach

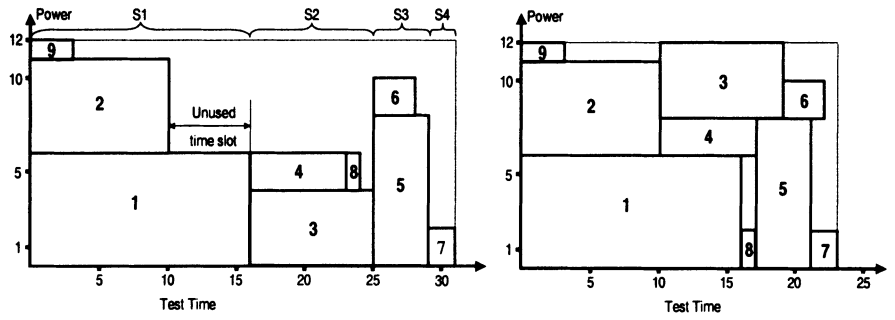

Figure 4. Scheduling results with 1/ unequal length test sessions [10] and 2/ no session

We also compare our technique with another method proposed in the literature targeting the no session based approach [11]. The authors solve the scheduling problem using a MILP formulation. They assume two test runs for every core, a BIST procedure in order to test most of the faults with a relatively low number of patterns and, when necessary, an external test procedure for detecting remaining hard-to-detect faults. Inequations are used for expressing conflicts and constraints between core tests. Note that the main drawback of the ILP formulation is the exponential growth in term of inequations. The authors report a total test time of 7985 cycles for the SoC example whose characteristics are given in Table 3.

We adapted our algorithm to support hybrid tests (BIST + external) so that the comparison with [11] can be possible. Since external test and BIST cannot be applied simultaneously on a core, the adaptation simply consists

to add incompatibility constraints between these two tests on every core. We applied this new version of our heuristic to the same SoC example with the same constraints on resource sharing limitations and power dissipation limit (Pmax=950 mW). Our solution is presented in Figure 5. Each bloc ii (resp. i) represents an external test (resp. BIST), for the core number i.

Core/number	BIST test time (cycles)	External test time (cycles)	Power dissipated in BIST mode (mW)
c880/1	256	134	54
c2670/2	2048	2543	159
c7552/3	2048	1357	453
s953/4	256	454	57
s5378/5	256	1903	324
s1196/6	256	242	72
s13207/7	2048	-	792
s1238/8	1024	176	75

Table 3: SoC example for a comparison with another NS-based approach

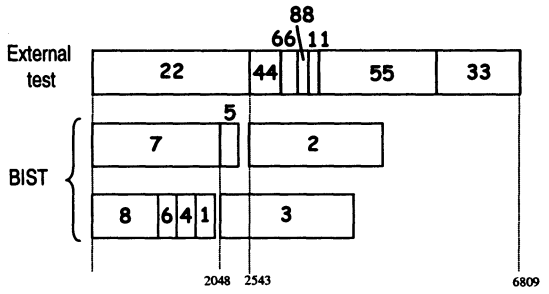

Figure 5. : Test scheduling for the SoC example described in Table 3

We obtained a total test time of 6809 cycles versus 7985. This example shows that in spite of its simplicity, our algorithm leads to good results.

Moreover, the CPU time is in the order of the second for about 100 cores in a SoC. Thus, we can foresee using our tool to explore different solutions. Figure 6 presents several scheduling solutions for the SoC example described in Table 1. As expected, the system test time decreases when the power dissipation constraint increases. The curve shows that different power dissipation limits lead to the same system test time, it also indicates the resulting increase in test time when the power constraint becomes stronger.

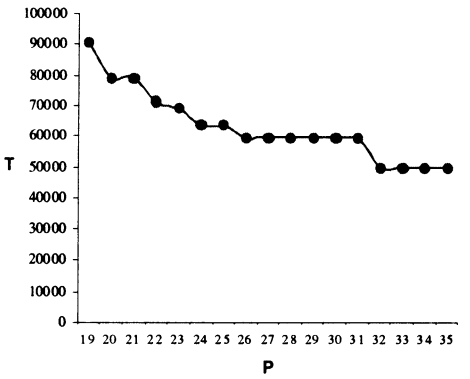

Figure 6. Exploration of the test scheduling solution space

6. EXTENSION

As mentioned above, the first extension for our tool was to adapt our algorithm in order to support hybrid tests during which several test runs are applied to every core (see secton 3.2). For instance, a core is first tested with pseudo-random patterns delivered by internal test resources, then tested with deterministic patterns delivered by the ATE.

As mentioned in [11] for an "abort at first fail" industrial test strategy, it is more convenient to run test processes that are more likely to detect defect first. This point introduces the notion of precedence in a test suite. In practice precedence constraints appear, if cores have to be tested first for many reasons like: they are bigger and have more chances to be faulty, they are purchased from external vendors and so tested before cores designed in-house. The last reason comes from the fact that core test sets can contain a BIST part and a deterministic part. The cores first tested should be the ones with the best rate (number of detected faults)/(test time), just to test the highest number of faults in the shortest possible time decreasing in this way the test time of a faulty core. Since the external test only targets few hard-to-detect faults in comparison with BIST that targets all other ones, BIST should be applied first.

Consequently the proposed algorithm has been modified in order to deals with precedence constraints. These user-given constraints concern any test and any core in a test suite. The function Place() in our algorithm (see section 3.1) takes into account these new constraints. The algorithm postpones a test if the ones that must be applied first have not been yet scheduled.

7. CONCLUSION

We have presented an efficient scheme for organizing the test at system-level and a corresponding power constrained test-scheduling algorithm. This approach outperforms classical ones, which are based on test sessions.

In spite of its simplicity, the proposed algorithm also outperforms other "no session" solution. Reasonable CPU times allow exploring a wide range of solutions. Proposed extensions allow to run several tests on the same core and to partially order tests on a test suite.

Finally, present work assumes that the test architecture is fixed before to solve the test scheduling problem. We expect a better test area and test time tradeoff by assigning dynamically the test resources when needed.

8. REFERENCES

[1] H.J. Wunderlich, "BIST for systems-on-a-chip", Integration, the VLSI journal, 26, pp 55-78, 1998.

[2] I. Ghosh, N. K. Jha, S. Dey, "A low overhead design for testability and test generation technique for core-based systems", Proc. Int. Test Conf., pp 50-59, 1997.

[3] E.J. Marinissen, R. Arendsen, G. Bos, H. Dingemanse, M. Lousberg, C. Wouters, "A structured and scalable mechanism for test access to embedded reusable cores", Proc. Int. Test Conf., pp 130-143, 1998.

[4] P Varma, S Bhatia, "A structured test re-use methodology core-based system chips", Proc. Int. test Conf., pp 294-302, 1998.

[5] K. Chakrabarty, "Design of system-on-a-chip test access architectures using integer linear programming", Proc. VLSI Test Symp., pp 127-134, 2000.

[6] R. Chou, K. Saluja, V. Agrawal, "Scheduling Tests for VLSI Systems under Power Constraints", IEEE Trans. on VLSI Systems, Vol. 5, No. 2, pp 175-185, June 1997.

[7] M.R. Garey, D. Jonhson: "Computers and Intractability: guide to the theory of NP-completeness", W.H. Freeman and Company, San Francisco.

[8] C.P. Ravikumar, A. Verma, G. Chandra, "A Polynomial-Time Algorithm for Power Constrained Testing of Core Based Systems" , ATS 99, pp 107-112.

[9] V. Muresan, X. Wang, M. Vladutiu, "List scheduling and Tree Growing Technique in Power-Constrained Block-Test Scheduling", ETW 99, pp 27-32.

[10] V. Muresan, X. Wang, M. Vladutiu, "A comparison of classical Scheduling Approaches in Power-Constrained Block-Test Scheduling ", ITC 2000, pp 882-891.

[11] V.Yengar, K. Chakrabarty, "Precedence-Based, Preemptive and Power-Constrained Test Scheduling for System-on-a-Chip", Proc. VTS '01, pp:368-374.