Simultaneous Module Selection and Scheduling for Power-Constrained Testing of Core Based Systems

C.P. Ravikumar
Dept. of Elec. Engg.
IIT Delhi
rkumar@ee.iitd.ernet.in

Gaurav Chandra
Synopsys India Ltd.
Bangalore
gauravc@synopsys.com

Ashutosh Verma *
IKOS India Ltd
Noida, India
ashutosh@ikos.com

Abstract

We address the problem of power-constrained testing of core based system chips. Built-in self-test methodology for testing individual cores is assumed, and sharing of test resources (pattern generators and signature registers) among cores is permitted. We consider a scenario where the system integrator is dealing with "soft" or "firm cores" for which the final realization has not been frozen and the flexibility of module selection rests with the integrator. We argue that advantage can be taken of this flexibility in coming up with a power-constrained test plan. Since scheduling of test sessions also affects power dissipation in a crucial way, we present an algorithm for simultaneous module selection and test scheduling. Our objective is to minimize the test application time treating the test area overhead and total power dissipation as constraints. We report thye results of our implementation of a test planner on two examples.

1 Introduction

It is now a well recognized fact that testing system chips is a hard problem since it must address many issues such as test reuse, test architecture, test expansion, and test power. We focus on the issue of power-constrained test planning. Since cores can be sizeable in complexity, each core can dissipate significant amount of power even in normal operation. In the test mode, since the inputs are applied in an uncorrelated manner, power dissipation of individual cores can be as high as 3 to 4 times the normal power. The total test power of the SOC can far exceed the power rating of the package.

In this paper, a *test plan* refers to the scheduling of tests and test resource allocation for all the cores in the SOC. BIST is assumed as the test methodology for all cores. We do not make any assumptions on the test architecture (e.g. isolation ring, etc.) since this choice does not majorly impact the test plan generated by our planner. Unlike existing literature on system testing [1, 4], we introduce a new dimension to the problem by permitting module selection for cores. Often, a system designer has several choices for realizing the same "soft" or "firm" core. While the normal performance of the system may not be affected by the choice of technology for individual cores, the tradeoffs in test power, time, and area may be considerable. In one of our examples, we demonstrate a saving of 35% in test time by technology selection. When the system consists of both firm and hard cores, we argue that technology selection and test scheduling must be considered simultaneously in order to obtain the maximum saving in test time. We formulate the problem as a constrained optimization problem and present a heuristic to solve it.

1.1 Previous Work

Lin, Njinda and Breuer [4] addressed the problem of generating testable data paths. They defined a test embedding as a tuple *(K, P, S),* where if is a "kernel" (combinational logic under test), *P* is a BILBO register [5] that will be configured as a pseudo-random test generator, and *s* is another BILBO register which will be configured as a signature register to test the kernel. Given the structural description of a data path, there are many embeddings for each kernel. One embedding is selected for each kernel. The choice influences the test application time and the test resource sharing. The authors formulated test scheduling as an optimization to minimize the test time or area overhead. A branch-and-bound algorithm was used to generate a family of designs that satisfy lower and upper bounds on the test time and the area overhead.

Chou, Saluja and Agrawal [1] have addressed the issue of power constrained test scheduling for application specific integrated circuits. The authors assume that the test resources have already been allocated. The authors assume that a *test compatibility graph* is given and derive an optimum schedule using a graph-theoretic optimization al-

* Gaurav Chandra and Ashutosh Verma were students at IIT Delhi when this work was carried out.

gorithm. A test compatibility *graphTCG(V, E)* is agraph whose node set *V* is the set of cores in the system and the edge set *E* consists of edges of the form (e^, *ej)* if and only if cores e^ and *ej* can be tested simultaneously under the test resource allocation. From the TCG, the authors derive a power compatibility graph PCG which is similar to the TCG, except that an edge (e;, *ej)* in the PCG implies that the cores *i* and *j* can be tested simultaneously without exceeding the power limit.

Note that the test resource allocation must be known *a priori* to apply the scheduling procedure described in [1]. None of the available literature takes into account the technology selection aspect which is addressed in this paper.

We assume the Built-in Self Test (BIST) methodology to test cores and consider a larger problem of determining the test resource allocation as well as scheduling the testing of cores. We use a "desirability matrix," a n n x n positive real matrix, *n* being the number of cores, which describes for every pair of cores *i* and *j* a quantitative measure *Wij* that measures the appropriateness of allocating the same test resources to test *i* and *j*. Using the desirability matrix, we derive a set of testable designs that respect both the power constraint and the area constraint. Each core must be allocated a pseudo-randompattern generators (PRPG) and a signature analyser (SA). The user can supply the information related to layout, similarities between cores, etc. so that the relative preferences of resource sharing can be evaluated.

2 Preliminaries

Testing "system chips" with embedded cores has received considerable attention in the recent past due to the growing popularity of design reuse. Among other problems, the issue of *test scheduling* must be resolved by the system integrator. Built-in self-test (BIST) has been employed in many system-on-chip designs [2, 7]. Here, the designer allocates extra test hardware to enhance the controllability and observability of the embedded cores, generate pseudo-random test patterns for testing the cores, and compact the responses of the cores. A number of possibilities exist, ranging from highly concurrent testing to entirely sequential testing of the cores. In the former situation, test resources such as pattern generators and response compactors cannot be shared among cores, leading to a large test area overhead. However, sharing of test resources among cores will lead to increased test application time since the cores will have to be tested in a sequential order. The scheduling problem is further complicated by the introduction of the test power constraint. The test power dissipation in a core may be several orders of magnitude higher than its rated power, since test vectors are applied in a random, uncorrelated manner. A high degree

Library	CORDIC		FPA	
	Power (*fiW*)	Length	Power (*fiW*)	length
mcnc	568.0	37	76.9	6
lib2	552.6	48	77.2	6
22-1	502.0	46	71.1	6
22-2	595.0	39	88.1	6
33-1	516.0	45	72.0	6
33-2	603.2	38	88.4	6

Table 1: Test power and time for different libraries

of concurrency in testing can thus lead to unacceptable amount of peak test power. Reducing the test clock rate can solve this problem, but leads to higher test application time.

If the system designer is provided with a behavioural level description of a core, often there might be various technology mapping options that offer trade-offs in test time and test power for a core. As an example, in Table 1, we present the test times and test powers for different technology mapping libraries of components CORDIC and FPA, used in a DCT computing system, described later in this paper. These libraries correspond to different gate level mappings, and possible trade-offs in test time and test powers are evident. Thus, ideally the test scheduler should explore all the options and choose the one leading to minimum overall test time.

2.1 Problem formulation

Given a set of cores $\{C_1, C_2, \cdots, C_n\}$ with test times $\{T_1, T_2, \cdots, T_n\}$ and test powers $\{P_1, P_2, \cdots, P_n\}$, we are **required** to partition the set of cores into a number of clusters $S_1, S_2, \cdots, S_k$, such that all the cores in a cluster S_j are tested concurrently. We refer to these clusters as test sessions. The length of a session j is the time taken to test all the cores assigned to S_j. Thus, $Length_j = \max_{i \in S_j} T_i$. The peak power dissipation during session j may be estimated as $Peak_j = \sum_{C_i \in S_j} P_i$. The objective of the scheduling problem can now be stated as one of minimizing $\sum_j Length_j$ such that the maximum value of $Peak_j$ does not exceed a specified value P_{max}. In case a number of mappings are present for a core, a number of schedulings are posible and mappings leading to miminum total test length are to be chosen.

In order to factor in the test area overhead and other issues such as similarity of cores into the problem, we use the notion of *desirability*. The desirability of using the same BIST resources to test two cores C_i and C_j is quantified by a positive real number W_{ij}. The larger the value of W_{ij}, the more desirable it is to share BIST resources for testing C_i and C_j. We shall explain the computation of W_{ij} later in the paper. We compute the *area penalty* of the

solution with the knowledge of the desirability matrix *W*, as we shall elaborate later. In the formulation of the power constraint, we have taken the peak power of a test session as the scalar sum of the peak powers of cores tested in that session. This is really an upper bound on the peak power dissipation, since individual peaks may actually be spaced out. We now discuss the formulation of the area constraint.

2.2 Desirability Graph

Unlike previous authors [1, 7] we do not assume a test resource allocation is specified a priori. We believe that it is easier for a system integrator to implicitly specify some guidelines about test resource allocation. For example, a system integrator may consider the functional similarities of cores (e.g. two embedded RAMs of similar size) and indicate that test resources can be shared among these cores. Apart from functional similarities, layout considerations can influence the integrator's decision to share test resources among cores. For instance, if two cores are physically far apart in the floor plan of the chip, it may be unwise to share the BIST resources. If there is a significant disparity between the test times/test powers of two cores, sharing test resources between the cores is again an unwise decision.

Let $W = [Wij]$ be an n x n matrix where entry Wy, i ^ j, indicates the desirability of sharing test resources for cores Cj and *Cj*. We interpret *Wu* as the cost associated with testing the core CV If the scheduler places two cores *Ci* and *Cj* in the same session *Su*, it is clear that test resources are not shared between Cj and *Cj*. We associate an "area penalty" resulting from this situation. The penalty associated with a test session *Su* is given by

$$AreaPenalty(Sk) = \sum_{C_i \in Sk} \sum_{C_j \in Sk} Wy - \lambda \sum_{C_i \in S_k} \sum_{C_j \notin S_k} w_{ts}$$

A is a constant in the range [0,1]. The area penalty for the complete schedule is defined as

$$AreaPenalty = max_k AreaPenalty(Sk)$$

The scheduling algorithm which we present in this paper attempts to keep the overall area penalty within a specified limit. The desirability Wy of allocating the same test resources for cores i and j is expressed as weighted sum of the following (a) similarity index Sy, (b) absolute difference of the test times T_i and Tj, (c) absolute difference of the test powers Pi and Pj, and (d) center-to-center distance dij between the cores i and j in the floor plan of the chip.

$$W_{ij} = a \cdot Sij - 0|Ti - T_j| - 7|P_i - Pj| - 5 \cdot dij$$

a, P, 7, and *S* are positive real constants indicating the relative importance of the four factors mentioned above.

3 Scheduling Algorithm

A *partial solution* **of level** i **is a set of** i **test sessions, such that the test scheduling has been done till that level. A number of partial solutions may exist at any level. For each partial solution, we have a test time, and a future set, which reflect the "worth" of the solution.** ***The** future set* **of a partial solution is the set of all cores not present in the partial solution. Thus, it contains in decreasing order of test time all the cores that have not been scheduled.**

```
Procedure MapperScheduler(CoreZist)
{
DescendingSort Corelist on average test times
// Test time is averaged over the number of mappings
// for the purpose of sorting.
R = $, Solution obtained = False
// R is the set of all possible partial solutions
for (all mappings Mi of C\)
{
    PSi = Construct_session(Ci, Mi)
    // Constructsession makes a test session
    // starting with a mapping of a core
    Add PSi to R
    // PSi is a first level partial solution
}
Prune(R)
// Getting rid of non-optimal partial solutions
While (Solution obtained = False)
{
        Expand each partial solution of R
        Prune(R)
        If (no element in R can be further expanded)
        Solution obtained = true
}
}
```

The subprocedures are now given below.

```
Procedure Constructsession(Core, Mapping)
{
Session <— {Core}
while (not end of corelist)
{
        Pick up the next unmarked element Cj from the list
        If (Cj has more than one mapping)
        Pick up the minimum power mapping
        // Time governed by 1 st core and penalties are same
        If (constraints not violated)
        Add Cj to Session
}
Return( Session)
}
```

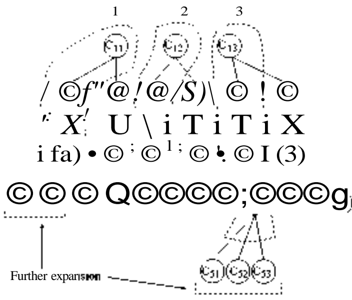

Figure 1: Illustrative example for the algorithm

Procedure Prune(R)

{
For all pairs of partial solutions *PSi* and *PSj* £ R
If ((future set is identical) & *(PSi* takes more time than *PSj*))
Delete $PS_{(}$
}

Procedure Expand(Partialsolution)

{
Pickup first core *Cj* from future set of partial so lns
For (all mappings M; of *Cj*)
{
  S = Constructsession(Cj, Mi)
  Append S to PartiaLsolution to get new partial soln
}
  // Many partial solns at next level are obtained
}

We illustrate the working of the algorithm by an example. See Figure 1. There are 3 mappings for core *C*, 2 for C_2, 1 for C_3, 2 for C_4, 3 for C_5. The algorithm considers all 3 mappings of *C*. Since the test time of the session is determined by *C*, the minimum power mapping of C_2, viz C21, is selected. Suppose the sessions corresponding to Cn and Cy_2 are "full" the session with C13 allows one more core, and *Cn* (with minimum power) is selected, completing this test session. At this stage, all the good partial solutions of level 1 are extracted and stored in *R*. Now, the first 2 partial solutions have the same future set and any expansion at next level will lead to similar test sessions. Hence the one having a larger test session will be pruned. Nothing can be said about the rest two as yet. The algorithm will proceed by expanding the two solutions. In case of first solution, two mappings of core C_4 will be considered and second test session will be created just like first session , leading to two partial solutions of second level. Similarly all 3 mappings of C_5 will be considered while expanding second partial solution, leading to 3 partial solutions of second level. The set R will now contain 5 partial solutions of second level. Some of these will again be pruned based on their future sets. This will go on till optimum solution is obtained.

We argue that as long as the assumption about the unique sorting holds, for any given ordering of cores in *Corelist,* the algorithm generates an optimum test schedule. For this, first we prove that the greedy subprocedure "Constructsession" leads to optimum schedule. Let *PowerSlack(j)* denote the slack in the power for session *j*. In other words, if *Pj* is the actual power dissipation for session *j* and P_{max} is the power limit specified, then $Power\ Slack(j) = P_{max} - Pj- AreaSlack(j)$ can be similarly defined. The first core *C* that begins the session Sj_{+i} would have been placed in session *Sj* if adding it to the session does not violate the area constraint and $Pc < PowerSlack(j)$. We now argue that the sorted order found in the first step of the scheduling algorithm results in the least test-time among all possible orders. To appreciate this, consider the cores are ordered as $d, C_2, C'_A, \bullet\bullet\bullet, C_n$. Further suppose that $T_2 < T_x$ and $T_2 < T_3$. Let T_3 be largest among $C_3, C_4, \bullet\bullet\bullet, C_n$. If *C* and C_2 are placed in the same session and C_3 is in a second test session (due to violation of constraints), we see that the total test time is equal to $T\backslash + T_3 + t$, where *t* represents the total time of sessions 3 and higher. If we now consider the order $C\backslash, C_3, C_2, C_4, \bullet\bullet\bullet, C_n$, we have the possibility of grouping *C* and C_3 together in one session. This results in a total test time of $Ti + T_2 + t$. However, our earlier assumptions imply that $(Ti + T_2) < (Ti + T_3)$. Based on the above observations, we can argue that "Constructsession" leads to optimum test scheduling for a particular mapping. To obtain the best mapping possible, the algorithm performs an exhaustive search over the solution space, thus leading to the best solution. Thus, we can argue that the scheduling algorithm presented above is optimal.

Since the algorithm has to search exhaustively for an optimum solution, the worst case time complexity is exponential, but it must be noted here that the pruning strategy applied at each level greatly reduces the actual number of cases. Moreover, since a number of mappings are considered only for the first core in a test session, as explained, the algorithm has to search through only a limited number of cases.

Since the number of cores in a system chip is in the range of 10 to 20, the actual execution time of the algorithm is not an issue on modern-day computers. In the examples that we tested, *n* was around 10, and the run time on a SPARC Ultra/1 was a few milliseconds.

4 Implementation and results

The scheduling algorithm described in the previous section was implemented using about 2000 lines of C code on a Sun Ultra/1 workstation. We used SIS [6], POSE [3], and a locally written RAM power estimation software tool for

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

Table 2: Power Dissipations and Test Lengths for ASIC Z

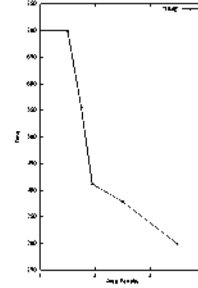

Figure 2: Test-time Vs Area Penalty for ASIC Z

Block	*Pactive(mW)*	Test length
RAM1	70	**5**
RAM2	70	**5**
ROM1	60	**5**
ROM2	60	**5**
GLUE	40	**4**

Table 3: Power Dissipations and Test Lengths for DCT

Ses	Cores	**Len**	Ses	Cores	Len
1	RL1,RL2	160	1	RL1, RL2	160
2	ROM1,ROM2 RAM1	102	2	ROM1, ROM2 RAM2	102
3	RAM2,RAM3 RAM4	61	3	RAM1, RAM3 RAM4, RF	69
4	RF	10			
Total		333			331

Table 4: Comparison with the work reported by Chou.

generating test vectors for individual cores and estimating core test power dissipation. We tested our software against two examples. The first example is the ASIC *Z* circuit described by Zorian [7]. ASIC *Z* has 4 RAMs, 2 ROMs, a register file, and three random logic blocks, all of which are treated as cores (see [7]). In the figure, BS-TAP is the boundary scan test access port. Since Chou et al. [1] present results for the ASIC Z example, we compare our results with those reported in [1]. The test power dissipation and the test lengths for each block in ASIC Z are given in Table 2.

To illustrate the benefit of technology mapping, we show an example (DCT) which uses 2 embedded RAMs, 2 ROMs, 2 CORDIC cores, a fixed point adder, a scaling unit, and glue logic. We have the behavioural descriptions for CORDIC and FPA and a range of technology mappings. The test time and test power estimates for these components were extracted using SIS [6] and POSE [3]. RefertoTable3.

The layout information for the cores is not given in [7] for the ASIC-Z example. For the purpose of our study, we assume that the block diagram is reflective of the floor plan of the ASIC. A floor plan was manually derived for the DCT example and used in determining the distances between cores. Similarity coefficients were derived based on the knowledge of the functionalities of the cores. For example, the similarity index for two embedded RAM cores is high (close to 1), whereas the similarity index for dissimilar blocks (say a RAM and a random logic block) is low (around 0.5).

For the ASIC Z example, we used a power constraint of $Pmax = 900$mW, which is same reported in [1] and [7]. Our software was able to report a large number of test schedules that meet the power constraint. These points are plotted in Figure 2, which clearly brings out the tradeoffs involved in test-time and test-area. As we increase the area penalty limit, we obtain more concurrent test sessions, leading to lower test times.

The test schedule obtained for a power limit of 900mW and area penalty limit of 3.5 is shown in Table 4. The schedule of Table 4 is compared with the schedule reported in [1]. Although the schedule obtained by [1] has a marginally (0.6%) lower test time compared to the schedule which we obtained, we note that the maximum concurrency level in our solution is 3, leading to lesser test area overhead. To illustrate that our technique can generate a number of solutions, we show the results obtained when the area penalty limit was increased to 4.0 (Table 5). We see that the number of sessions has reduced to 3 and the test time has reduced to 300, which is 10% lower than the result reported in [1].

The results obtained for DCT are in Table 6. We show results for two area penalty limits, 2.0 and 8.0. When the lower area penalty was specified, and the power limit was set to 1mW, the solution has 4 test sessions, a test length of 84, and the peak power dissipation is 645 /zW. Based on this feedback, we raised the area penalty to 8.0 and reduced the power limit to 700 /zW. The resulting solution

Session	Cores Tested	Session Length
1	RL1,RL2,RAM2	160
2	ROM1,ROM2,RAM1	102
3	RAM3,RAM4,RF	38
Total		300

Table 5: Solution for ASIC Z example; Area Limit = 4.0, Power Limit = 900 mW.

Ses	Cores	**Len**	Ses	**Cores**	**Len**
1	FPA, CORDIC1	37	1	GLUE, FPA CORDIC1	37
2	RAM1 CORDIC2	37	2	ROM1, RAM1 CORDIC2	37 37
3	ROM1,RAM2	5	3	ROM2, RAM2	5
4	GLUE,ROM2	5			
Total		84			79

Table 6: Solutions for DCT Example

has 3 test sessions, a total test length of 79, and a peak power dissipation of 698 */JW*. Figure 3 shows the plot of design points obtained when the power limit was fixed at 900 */JW* and the area penalty was altered.

To emphasize the importance of technology mapping, we compare the solution space obtained for the DCT example for the case when only a single library was used and the case where multiple libraries were used. Figure 4 shows the design points for different power limits, but a fixed area penalty of 8.0. Improvements of up to 35 % may be seen.

5 Conclusions

We presented an approach to optimally schedule n embedded cores, that takes care of designer's concerns, selecting the best modules, and generates a design for testability solution. The desirability factor associated with a pair of cores i and j can be computed using the information about the similarities of cores and details of the floor plan of the chip. An interactive software tool can be built towards this end. Since the scheduling algorithm itself is fast, it is possible for a system integrator to interactively explore a variety of solutions by changing the desirability factors, providing new libraries and rescheduling the tests. The concepts presented in this paper can be easily extended to board-level designs and multi-chip modules.

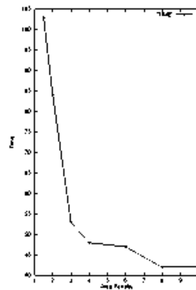

Figure 3: Test-time Vs Area Penalty for DCT

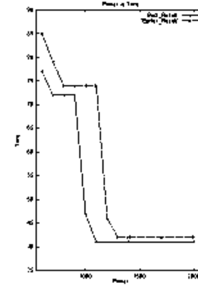

Figure 4: Improvements by Technology Mapping

References

[1] R.M. Chou, K.K. Saluja and V.D. Agrawal. *Scheduling Tests for VLSI Sys. Under Power Constr.* IEEE Tran. on VLSI Sys., 5(2), 175-185 1997.

[2] R.K. Gupta and Y. Zorian. *Inroducing Core Based Design* IEEE Design and Test of Computers, 15-25, Oct-Dec 1997.

[3] S. Iman and M. Pedram.Poww *Optimization and Synthesis Environment* Dept. of EE-Systems, U. of S. Calif. 1995.

[4] S.P.Lin, C. Njinda, and M.Breuer. *Generating a Family of Testable Designs using the BILBO Methodology.* JETTA, 71-89, 1993.

[5] B.Konemann, J.Mucha, and G.Zwiehoff. *Built in Loggic Block Observation Technique.¥mc.l\A.Test* Conf., 37-41, 1979.

[6] E.M.Sentovich et al. *A System for Sequential Circuit Synthesis.* Dept. of EECS, UC Berkeley. 1992.

[7] Y. Zorian. A distributed BIST control scheme for complex VLSI devices. Proc. of 11th IEEE VTS. 1993,4-9.