Static Pin Mapping and SOC Test Scheduling for Cores with Multiple Test Sets

Yu Huang[1] Wu-Tung Cheng[1] Chien-Chung Tsai[1] Nilanjan Mukherjee[1] Sudhakar M. Reddy[2]
[1] Mentor Graphics Corporation, 8005 S.W. Boeckman Rd., Wilsonville, OR 97070
[2] Department of Electrical & Computer Engineering, University of Iowa, Iowa City, IA 52242

Abstract

An algorithm for mapping core terminals to System-On-a-Chip (SOC) I/O pins and scheduling tests in order to achieve cost-efficient concurrent test for core-based designs is presented in this paper. In this work "static" pin mapping and test scheduling for concurrent testing are studied for the case of multiple test sets for each core. The problem is formulated as a constrained two-dimensional bin-packing problem. A heuristic algorithm is then proposed to determine a solution. The objectives driving this solution are geared towards reducing the total test application time of SOC and satisfying the test constraints such as limited number of SOC pins and maximum peak power dissipation specified by core integrators. Experimental results demonstrate the effectiveness of the proposed method.

1 Introduction

Core-based SOC design strategy is becoming a popular design methodology. The Semiconductor Industry Association's (SIA) Technology Roadmap [1] predicts the percentage of reusable cores in SOC to be rising to 80% in 2006, thereby resulting in a 50% reduction of time-to-market. However, conflicting design objectives such as increasing complexity and reduced design cycle lengths have made the test application time a major bottleneck towards achieving aggressive marketing requirements. Concurrent SOC testing (i.e. testing more than one core simultaneously) is becoming an attractive solution to reduce the total test application time under such circumstances. In this paper, a solution addressing the pin mapping and test scheduling problems in concurrent SOC testing is presented.

The paper is organized in the following manner. In Section 2, related work in the SOC test scheduling and pin mapping is reviewed. In Section 3, formulation of the pin mapping and test scheduling problem is presented, and in Section 4, a heuristic algorithm to achieve optimized concurrent SOC test is proposed. Experimental results are presented in Section 5, followed by the conclusions section.

2 Review of Related Work

The complexity of a SOC makes test a much more difficult problem. Many new DFT techniques have been exploited to address this problem. The SOC composite test requires effective test scheduling given a number of chip level requirements such as total test time, power dissipation, pin limitations etc. SOC test should be created to satisfy these scheduling constraints. Previous research in the area is discussed in the following paragraphs.

Chakrabarty [2] formulated the SOC test scheduling problem as an *m*-processor open shop scheduling problem, and solved it by using a Mixed-Integer Linear Programming (MILP) model in order to minimize the test time. This formulation is applicable to the test bus-based architectures, and assumes each core can only be assigned to one bus. Ravikumar et al. [3] proposed a method to solve SOC test scheduling problem under the constraint of power consumption. They assumed that BIST is the only methodology for testing individual cores.

Huang et al. [4] proposed using bin-packing to allocate test resources and schedule test sets in order to achieve optimal concurrent SOC test. In [4] it was assumed that each core has a single test set. The objective was to minimize the test application time for different Test Access Mechanism under the constraint of peak power consumption.

Compared with the above-mentioned methods, the novelty of the method proposed in this paper includes the considerations of the following:

(1) There may be more than one test set for each core. It may include scan structural testing, functional testing, on-chip memory testing, at-speed testing, Logic BIST, etc. [5]. In [2], only two sets of test (one external test set and one BIST session) are considered for each core. In this paper, however, the number of test sets is not limited.

(2) Each test set of a core may only be applied through a subset of the set of pins on the core. For example, the functional test will not use scan pins, and a BIST session can be controlled by a few pins such as "Reset", "Run", "Pass_Fail" signals etc. that are specific to BIST.

0-7695-1881-8/03 $17.00

The previous work assume that each external test set is applied through all core terminals and the SOC pins to access BIST controllers are ignored. This is another practical issue considered in this paper.

(3) For the case of single test set for each core, optimum test scheduling and mapping of pins of cores to the pins of SOC is to minimize the time to test all cores by mapping core pins onto SOC pins and determining the start of the test of each core. For the case of multiple test sets for cores, one needs to determine the optimum time to start each test set while taking into consideration that some core pins are used in multiple test sets for the core.

3 Problem Formulation

Problem: Given N SOC pins and K cores, for each core C_i ($0<i\leq K$), there are S_i test sets. A set of 4-tuples $\{(M_{i,1}^{in}, M_{i,1}^{out}, T_{i,1}, P_{i,1}), (M_{i,2}^{in}, M_{i,2}^{out}, T_{i,2}, P_{i,2})\ldots\ldots (M_{i,Si}^{in}, M_{i,Si}^{out}, T_{i,Si}, P_{i,Si})\}$ is created. Each 4-tuple corresponds to a test set of C_i. $M_{i,j}^{in}$ is the number of inputs of $C_{i,}$ which are to be mapped to SOC pins when applying j^{th} test set to $C_{i.}$ $M_{i,j}^{out}$ is the number of outputs of $C_{i,}$ which are to be mapped to SOC pins when applying j^{th} test set to $C_{i.}$ $T_{i,j}$ is the test application time of the j^{th} test set for C_i, $P_{i,j}$ is the power consumption when testing C_i using j^{th} test set. Ω is the maximum peak power that cannot be exceeded.

The objective is to determine a *static mapping* (defined below) from I/Os of each core C_i to SOC I/Os and a test schedule for each test set j of core C_i. The realization of the concurrent SOC test needs to satisfy the following conditions.

(1) Minimize total test application time.

(2) At any given time t, a set of cores (C_{t_1} $C_{t_2}\ldots C_{t_r}$) can be tested simultaneously. Let the test set applied to core C_{t_i} at time t be T_{t_i,j_t}. Then the total power consumption should satisfy the following constraint: $\sum_{i=1}^{r} P_{ti,\ jt} < \Omega$.

By *static mapping*, it is implied, that once a pin on a core is mapped to a SOC pin for applying a test set, the mapping relation cannot be changed when applying other test sets. It is a practical requirement from our customer based on the consideration of reducing the number of MUXes and routing area.

Claim: The above problem is NP-Hard.

Proof: Restrict the above problem such that for each core only one test set is required. This is the special case addressed previously in [4], and was shown to be NP-hard problem. Therefore the above problem is also a NP-hard problem.

4 Proposed Heuristic Algorithm

4.1 A core example

We first give an example to illustrate several concepts used in our method. Assume that all SOC pins can be transformed to be bi-directional. This assumption is mostly true in an industrial environment, as the requirement for all SOC pins to be bi-directional adds trivial and hence acceptable overhead to SOC design. However, this assumption allows us to ignore the distinction between input and output pins of cores. Let $M_{i,j}^{in} + M_{i,j}^{out} = M_{i,j}$, be the total number of pins to be mapped to SOC pins when applying test set j to core C_i. Each pin of core C_i is to be mapped to a SOC pin during the application of one or multiple test sets. If a pin on a core is not used for any test set, it is not considered for mapping in this problem. We divide the set of pins of a core C_i into W_i disjoint groups. In group k there are $P_{i,k}$ pins. These pins are used when applying a unique subset of the S_i test sets of core C_i. For each core C_i we construct a rectangular array R_i. R_i has S_i rows with each row j having a height of $T_{i,j}$ corresponding to the test application time of the *jth* test set to core C_i. R_i has W_i columns. The width of the jth column is $P_{i,j}$, the number of pins in the jth group of the W_i groups defined above. Thus each core is represented by a rectangular array of S_i rows and W_i columns whose heights and widths, respectively, are equal to the application time of different test sets and the number of pins that are used in applying unique subsets of the set of tests.

For example, let C_1 be a core in a SOC with 3 test sets. (i.e. S_1=3). The 1st test set needs 2000 clock cycles and the pins used when applying the 1st test set are in the range [1,170]. (i.e. $M_{1,1}$ = 170, $T_{1,1}$ =2000). The 2nd test set needs 1000 clock cycles and the pins used when applying the 2nd test set are in the range [1,220] (i.e. $M_{1,2}$ = 220, $T_{1,2}$ =1000). The 3rd test set needs 800 clock cycles and uses pins in the range [1,100] AND [171,220] (i.e. $M_{1,3}$ = 100+50=150, $T_{1,3}$ =800). Therefore the total number of pins on the core C_1 is 220. These pins are divided into 3 (W_1 =3) disjoint groups. In group *1*, there are 100 pins ($P_{1,1}$ = 100) that are used when applying test sets 1, 2 and 3. In group *2*, there are 70 pins ($P_{1,2}$ = 70) that are used when applying test sets 1 and 2. In group *3*, there are 50 pins ($P_{1,3}$ = 50) that are used when applying test sets 2 and 3.

The rectangular array R_1 corresponding to core C_1 has 3 rows (S_1=3), and 3 columns (W_1 = 3), and 7 sub-rectangles as shown in Figure 1.

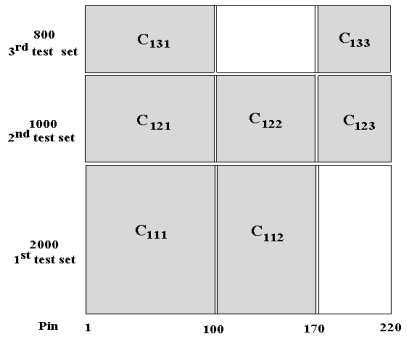

Figure 1: A core example

Note that:

(1) R_i is composed of an array of sub rectangular blocks $C_{i\,j\,k}$, if the j^{th} row k^{th} column is not a blank area (i.e. the pins in k^{th} column are used when applying the j^{th} test set). The above array may contain empty or blank spaces.

(2) In concurrent scheduling and pin mapping, the rows of sub-rectangles can be separated since it is not necessary to apply all the test sets consecutively. However, tests within each test set of a core must be applied consecutively.

(3) The order of the test sets applied for each core can be changed if not specified by users. In this paper, the proposed algorithm will determine the optimal order. If the user specifies the order, it could be easily added as a constraint, which is not considered in this paper. However, when the order is changed the entire row of the sub-rectangles is to change together.

(4) The order of columns can be changed (this depends on the pins of the core mapped to the SOC pins) and the columns can be separated. However, when the order is changed or separated the entire column of the sub-rectangles is to change or separate together because we only consider static mapping.

(5) The order of pins inside a column is not important.

4.2 Algorithm Explained with an SOC example

In order to give a general introduction on how we solve the problem and clarify the observations made above, we use an example illustrated in Figure 2. In this SOC example there are two cores with a total of 420 pins and the SOC has a total of 400 pins.

Pre-Processing: Construct the rectangular arrays for each core as described above. These are shown in the Figure 2 for the example we are considering. Next construct an ordered list of sub-rectangles. The order of the list is determined as follows.

(1) Sort all the sub-rectangles in the descending order of the height of the rows containing them, keeping all sub-rectangles in the same row in the order they appear in the correponding rectangular array.

(2) If two or more rows have the same height, order the sub-rectangles in the descending order of the total width of the sub-rectangle in the row.

(3) Order the sub-rectangles within a row in the descending order of the width of each sub-rectangle.

For the example with 2 cores in Figure 2, the ordered list of sub-rectangles is L={C_{211}, C_{212}, C_{111}, C_{112}, C_{121}, C_{122}, C_{123}, C_{131}, C_{133}, C_{221}}.

After the above pre-processing step, what we need to do is to pack all sub-rectangles $C_{i\,j\,k}$ in a bin with fixed width (number of SOC pins is fixed). It can be formulated as a two-dimensional bin-packing problem with several additional constraints. The original un-constrained "best-fit" algorithm for solving two-dimensional bin-packing problem [4] is briefly reviewed below.

The "best-fit" algorithm is a level-based packing algorithm. First, all the rectangles are put into a list and sorted in descending order of their heights. Second, rectangles are picked out of the list in the order they are arranged and placed in a bin with fixed width one after another. It is like building a book shelf level by level with minimum height to hold all the books. The height of each level is determined by the first rectangle placed in the level, and it is placed left-most at this level. Next we attempt to place the unplaced rectangles in the existing levels at the left-most available positions. A new level will be added if a rectangle cannot be placed on any existing level. If a rectangle can be placed on more than one level, the rectangle will be placed in its "best-fit" level, which means the unoccupied width at this level is minimum if the rectangle is placed in this level. The process terminates when the list of rectangles is empty. During the packing, the peak power constraint is satisfied at all time.

The optimal test application time schedule for an SOC can be obtained by using the above "best-fit" method with modifications to accommodate the following new constraints:

(1) If two sub-rectangles of a core are in the same row before packing, they remain in the same row after packing.
(2) If two sub-rectangles of a core are in the same column before packing, they remain in the same column after packing.

These constraints are satisfied in our method by a mechanism called *Reservation*.

There are two kinds of reservations: *Row Reservation* and *Column Reservation*:
(1) *Row Reservation:* If a sub-rectangle is placed in a level, then all the other sub-rectangles in the same row of the correponding core array R_i, have to be placed in the same level. A level number is reserved for them.
(2) *Column Reservation*: If a sub-rectangle is placed in a column, then all the other sub-rectangles in the same column of the correponding core array R_i, have to be placed in that column but on different levels. A set of pins of the SOC are reserved for them.

Let Row_{ij} represents the jth row of the rectangular array R_i and Col_{ij} represents the jth column of R_i. $|Row_{ij}|$ is the width of the row which is the number of pins used to apply the corresponding test set. It is the total width of all the sub-rectangles in the jth row of R_i. In the example shown in Figure 2.

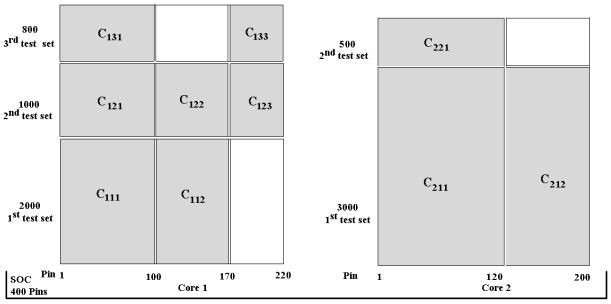

Figure 2: A SOC example.

For Core 1, there are 3 Rows:
$Row_{11} = \{C_{111}, C_{112}\}$, $|Row_{11}| = 100+70 = 170$;
$Row_{12} = \{C_{121}, C_{122}, C_{123}\}$,
$|Row_{12}| = 100+70+50 = 220$;
$Row_{13} = \{C_{131}, C_{133}\}$, $|Row_{13}| = 100+50 = 150$;

For Core 1, there are 3 Columns:
$Col_{11} = \{C_{111}, C_{121}, C_{131}\}$, $Col_{12} = \{C_{112}, C_{122}\}$, $Col_{13} = \{C_{123}, C_{133}\}$.

For Core 2, there are 2 Rows:
$Row_{21} = \{C_{211}, C_{212}\}$, $|Row_{21}| = 120+80 = 200$;
$Row_{22} = \{C_{221}\}$, $|Row_{22}| = 120$;

For Core 2, there are 2 Columns: $Col_{21} = \{C_{211}, C_{221}\}$, $Col_{22} = \{C_{212}\}$;

The difference between the modified "best-fit" algorithm and the orginal "best-fit" algorithm is as follows.

(1) Each time a sub-rectangle is placed, the unplaced sub-rectangles in the same row or the same column have to make reservations. The row reservation has to be maintained. However, the column reservation may be cancelled on certain levels if its position is occupied by some other sub-rectangles on those levels. It is based on " first come first served" policy.
(2) When checking whether a sub-rectangle could be placed on a certain level, it has to consider the width of its row rather than the width of the sub-rectangle only.
(3) When a sub-rectangle to be placed has a reserved row it has to be placed on that row. The columns chosen to hold the sub-rectangle should lead to minimum number of pins that have been column reserved for other sub-rectangles.
(4) When a sub-rectangle to be placed has reserved columns it has to be placed on these columns. The level chosen to hold the sub-rectangle should lead to minimum number of unused pins at this level.
(5) If a sub-rectangle is column reserved first and later row reserved, its position is fixed. It has to be packed at those columns on that level, which is called *position implication* (Note that, there is NO situation where a sub-block is row reserved first, and it is column reserved later. This is because in the pre-processing stage, the order of the list of sub-rectangles guarantees that all the sub-rectangles in the same row will be placed before processing sub-blocks in another row). In this case, we need to check whether the position implication is valid or not. If that position has already been occupied, we need to undo the row reservation, and undo the placement of the sub-block which leads to the row reservation. Then we have to search other levels. A new level has to be built if it fails for all the existing levels.
(6) The height of each level is not determined by the left-most sub-rectangle on this level, but it is determined by the height of the first sub-rectangle placed on this level. The order of sub-rectangles being processed may not be always according to the order in the list L due to position implication.

For the previous example given in Figure 2, the procedure is illustrated below.
(1) Pack C_{211} on level = 0; mark **Row-Reserved** for C_{212}, mark **Col-Reserved** for C_{221}.
(2) Pack C_{212} on level = 0;
(3) Pack C_{111} on level = 0; mark **Row-Reserved** for C_{112}, mark **Col-Reserved** for C_{121}, C_{131}.
(4) Pack C_{112} on level = 0; mark **Col-Reserved** for C_{122}.

(5) Next, C_{121} cannot be packed on level 0, Pack C_{121} on level = 1, mark **Row-Reserved** for C_{122} and C_{123}. Perform **Position-Implication**: C_{122} has 2 marks (**Col-Reserved** and **Row-Reserved**), so its position is fixed. Pack C_{122} at the reserved postion.
(6) Next Pack C_{123} on level = 1; mark **Col-Reserved** for C_{133}.
(7) Pack C_{131} on level = 2; mark **Row-Reserved** for C_{133}. **Perform Position-Implication**: C_{133} has 2 marks (**Col-Reserved** and **Row-Reserved**), so its position is fixed. Pack C_{133} at the reserved postion.
(8) Pack C_{221} on level = 1, in the reserved area.

The final packing is illustrated in Figure 3.

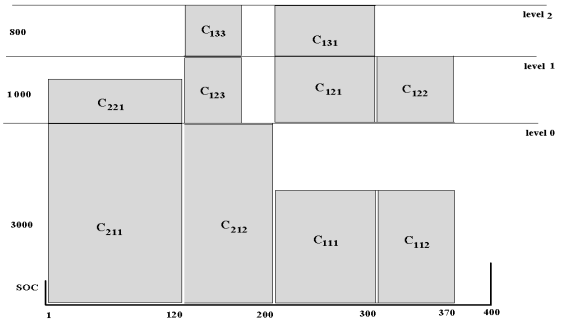

Figure 3: Final Solution for the example

By packing the two cores together, we can achieve concurrent core testing, and the total test application time is 3000+1000+800 = 4800 cycles. If the two cores are tested sequentially, the test application time is 3800 + 3500 = 7300 cycles.

The reservation mechanism is used to constrain some specific sub-rectangles to be packed in required rows or columns. The proposed algorithm is called "constraint-driven 2-Dimensional Bin-Packing".

5 Experimental Results

In order to evaluate the proposed algorithm, we compare the test application time for concurrent and sequential SOC test. There are a total of 10 ITC'02 SOC benchmarks [7] at the time this paper was written. None of them have multiple test sets for each core. Therefore, the algorithm is run on 10 hypothetical but nontrivial SOCs. The algorithm was implemented in C running on a SUNBlade1000 workstation. The CPU time for each benchmark is only in the order of seconds. That's why the CPU time is not included in the tabulated results.

Table 1 shows the information on 10 cores. We use 10 ITC'99 benchmarks as 10 hypothetical cores. Each core has 3 test sets: full scan structural test, functional test and BIST. The test length and number of pins needed for each test set are listed in Table 1. The test length for full scan structural test is obtained by running a commercial ATPG tool. The test length for functional test was given in [6]. The BIST application cycles are generated randomly. Five BIST controller pins are assumed for each core. Table 2 shows the information on 10 SOCs. Each of them is composed of some cores in Table 1. The cores may not be included in a SOC or could be included multiple times as indicated in Table 2. The total number of cores in each SOC is given in the last column of Table 2. The power consumption of the cores are unknown and hence ignored in the experiments. However, power constraint can be accommodated in the algorithm if known.

The experimental results of applying the proposed procedure are given in Table 3. We assume that the SOC has 200 pins. The ratio, between the test application time when concurrent SOC test achieved by the proposed procedure and the test application time when each core is tested sequentially, is given in Table 3. From Table 3, it can be seen that the proposed algorithm could, on average, save 78.9% SOC test time. In order to compare with the unconstrained 2-D bin-packing method in [4], we make one large rectangle out of all sub-rectangles that are on the same core, and pack these rectangles based on the best-fit algorithm [4]. The test cycles computed by the two algorithms are given in Table 4. From the experimental results, it is obvious that we can save 43.8% SOC test time on average compared to the method proposed in [4]. This is because the proposed method considers allocating test resource and scheduling test based on individual test set instead of all tests for a core. The unused test resource for one core when applying one test set can be utilized by another core, which is not feasible in [4].

Table 1: Information on cores

Benchmarks	Structural Test		Functional Test		BIST	
	Test cycles	# of pins used	Test cycles	# of pins used	Test cycles	# of pins used
B01	92	10	152	6	128	5
B02	66	8	42	4	64	5
B03	432	18	246	10	256	5
B04	1035	39	52	21	1024	5
B06	138	14	74	10	128	5
B07	609	29	168	11	512	5
B08	336	29	310	15	512	5
B09	177	24	236	4	256	5
B10	237	37	159	19	256	5
B11	636	145	542	131	1024	5

Table 2: Information on Randomly Created SOCs

Composition of SOC	B01	B02	B03	B04	B06	B07	B08	B09	B10	B11	# of cores
S1	0	0	1	1	1	0	1	0	1	0	5
S2	1	2	0	3	1	1	0	1	1	0	10
S3	3	1	2	0	2	2	1	3	0	1	15
S4	3	1	2	2	0	3	2	1	4	2	20
S5	2	1	4	3	4	3	2	1	5	0	25
S6	3	4	2	4	2	0	3	4	3	5	30
S7	5	2	4	3	5	4	1	2	4	5	35
S8	0	6	4	7	5	6	3	5	2	2	40
S9	6	5	3	3	4	5	6	1	4	8	45
S10	2	5	4	3	8	4	5	6	7	6	50

Table 3: Comparison of Concurrent SOC and Sequential SOC test.

SOC	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	Avg.
Ratio(%)	42.46	20.52	20.31	20.46	11.63	22.86	19.45	13.22	21.75	18.26	21.09

Table 4: Comparison of the Proposed SOC Scheduling and the Method in [4].

SOC	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	Avg.
Test Cycles by Proposed Alg.	2369	2236	2933	4857	3007	7616	7732	6204	11442	9448	5784
Test Cycles by Alg. In [4]	2111	2763	4873	8215	5433	13305	14332	12673	20852	18344	10290

6 Conclusions

In this paper, an algorithm to schedule tests and map pins efficiently for achieving concurrent SOC test under specific test constraints is presented. The objective is to minimize the total test application time for SOC, and consider multiple test sets for each core. A heuristic algorithm is proposed to solve the problem based on the constraint-driven 2-Dimensional Bin-Packing algorithm. Experimental results show that the proposed method is effective in reducing the total test application time for core-based design and is better than a previous method in this field.

References

[1] Semiconductor Industry Association, The National Technology Roadmap for Semiconductors, Sematech, Inc. 1997.

[2] K. Chakrabarty, “Test Scheduling for Core-Based Systems Using Mixed-Integer Linear Programming,” pp.1163-1174, IEEE TCAD, Vol. 19, Oct., 2000.

[3] C. P. Ravikumar, G. Chandra, and A. Verma, “Simultaneous Module Selection and Scheduling for Power-Constrained Testing of Core Based Systems,” VLSI Design 2000.

[4] Y. Huang, W.-T. Cheng, C.-C. Tsai, N. Mukherjee, O. Samman, Z. Yahya, and S. M. Reddy, “Resource Allocation and Test Scheduling for Concurrent Test of Core-Based SOC Design,” pp.265-270, ATS 2001.

[5] P. Harrod, “Test Reusable IP-A Case Study,” pp.493-498, ITC’99.

[6] F.Ferrandi, G. Ferrara, D. Sciuto, A. Fin and F. Fummi, “Functional Test Generation for Behaviorally Sequential Models,” pp.403-410, DATE2001.

[7] http://www.extra.research.philips.com/itc02socbenchm