An Efficient Approach to SoC Wrapper Design, TAM Configuration and Test Scheduling

Julien Pouget, Erik Larsson, Zebo Peng, Marie-Lise Flottes, Bruno Rouzeyre

An Efficient Approach to SoC Wrapper Design, TAM Configuration and Test Scheduling

Julien Pouget, Erik Larsson, Zebo Peng

Embedded Systems Laboratory
Linköping University, Sweden

Marie-Lise Flottes, Bruno Rouzeyre

LIRMM
Montpellier 2 University, France

Abstract

Test application time and core accessibility are two major issues in System-On-Chip (SOC) testing. The test application time must be minimised, and a test access mechanism (TAM) must be developed to transport test data to and from the cores. In this paper we present an approach to design a test interface (wrapper) at core level taking into account the P1500 restrictions, and to design a TAM architecture and its associated test schedule using a fast and efficient heuristic. A useful and new feature of our approach is that it supports also the testing of interconnections while considering power dissipation, test conflicts and precedence constraints. Another feature of our approach is that the TAM is designed with a central bus architecture, which is a generalisation of the TestBus architecture. The advantages and drawbacks of our approach are discussed, and the proposed architecture and heuristic are validated with experiments.

1. Introduction

Recent advances in IC design methods and manufacturing technologies have led to the integration of a complete system onto a single IC, called system on chip (SOC). These *system chips* offer advantages such as higher performances, lower power consumption, and decreased size and weight, when compared to their traditional multichip equivalents. Many system chips are designed by embedding large reusable building blocks, commonly called *cores*. Such a design reuse approach speeds up the design process, and allows import of external design expertise. However, the increased system complexity leads to high test data volumes, which means long testing times [Harr99]. Furthermore, traditionally, the chips are tested before they are integrated into a system. The interconnections are tested separately in system test when fault-free chips are already integrated. For system chips, on the other hand, testing of cores and interconnections is performed in a single system test step. Test access becomes also a problem for system chips since the cores are not directly accessible via chip inputs/outputs.

Power consumption, interconnection tests, test conflicts and precedence relations are important issues that must be addressed during the design of the test schedule. The power dissipation during the test mode is often higher than that in system mode because of higher number of signal switches. Thus, it is necessary to organize the tests in such a way that power dissipation does not exceed a given threshold. It is also important to test interconnections between cores. One problem here is test resource sharing since interconnections usually do not have a direct interface to the test data transportation mechanism. Therefore an important conflict that must be considered for interconnection test is the sharing of test resources. On the other hand, each core (e.g. hard core) may come with its dedicated BIST (Built-In Self Test) resource. The BIST resources can in some cases be shared between several cores, but this is not always the case. If they are shared, it is usually not possible to test more than one core at a time for a given resource. Furthermore, depending on the design, some tests may have to be applied before others. The order in which the tests are applied is therefore important, and precedence constraints must be considered.

In [MaGo00], the authors present two heuristics aiming at designing a wrapper for cores. The presented technique minimizes the longest wrapper chain for a given number of wrapper chains (i.e. the number of connections to the TAM). Iyengar and Chakrabarty [IyCh01], on the other hand, present a heuristic similar to the one we describe in this paper, which tries to minimize also the number of wrapper chains.

Once the wrapper is designed, a TAM must be built, and several approaches can be used. In [NoPa01], the authors use existing resources to implement the TAM. The approach searches, in an exhaustive way, all the controllable components connecting two points in the SoC (muxes, tristates, bypass, etc.). It deals with test scheduling and TAM architecture design simultaneously. In [CoCa02], the presented method uses different kinds of TAMs during the scheduling phase. The authors define a connection model of cores to neighbor cores with a cost for each of them. The aim of the approach is then to minimize this cost.

The more standardized approaches [IyCh01] [IyCh02] use specific test buses. The TestRail used by [IyCh01] [IyCh02a] mixes the Daisy Chained and the distributed architectures. Cores on the same TestRail are tested simultaneously. Another approach, called TestBus, is based on the mixture of the multiplexed and the distributed architectures. The cores connected to the same TestBus are tested sequentially, each core being connected to the whole TestBus bandwidth. The bus-based approaches are easy to implement and are more flexible than the other ones. They provide a direct compatibility with P1500 requirements [Poug02].

In this paper we propose a technique for the design of the wrappers, the selection of the TAM configurations and the scheduling of the tests. The main advantages of our approach are that we design a test schedule to minimize the test application time while considering power consumption and test conflicts. The conflicts we consider include test resource sharing and precedence relations. Furthermore, we consider also the testing of interconnections.

The rest of this paper is organised as follows. Section 2 presents our wrapper design algorithm, which generates a set of design alternatives. Section 3 describes the implementation of the TAM architecture and the heuristic we use in parallel to minimise the total test time. Section 4 summarises the experimental results and discusses the features of our approach. Finally, section 5 presents the conclusions and limitations of the proposed algorithms.

2. Wrapper Design

Before designing an overall test architecture for a given SOC, each core has to be wrapped considering the P1500 restrictions [MaIy02] for signals and functionalities. This interface allows us to isolate the cores during testing and to apply test vectors in an optimal way in terms of test time.

The purpose of our wrapper design algorithm is to develop a set of wrapper chains at each core. A wrapper chain includes a set of the scanned elements (scan-chains, wrapper input cells and wrapper output cells). The main objective of the algorithm is, for a given bandwidth, to organize the wrapper chains in such a way that the test time is minimized. The test time is related to the length of the wrapper chains, which means that we should minimise the longest wrapper chain (internal or external or both), *i.e.* ***max{si, so}***, where si (so) denotes the number of scan cycles required to load (unload) a test vector (test response). The test time at a core is given by:

$$T_{core} = p \times [1+\max\{si,so\}] + \min\{si,so\}$$

where p is the number of test vectors to apply to the core [IyCh02].

In our approach, a TestBus model for the TAM is used. The consequence of this is that we need connections to the TAM for the inputs and the outputs of the wrapper (Fig. 1). Our heuristic can be divided in two main parts; the first one for combinational cores and the second one for sequential cores. For combinational cores, there are two possibilities. If the TAM bandwidth limit, W, is above or equal to ***I+O*** (where ***I*** is the number of functional inputs and ***O*** the number of functional outputs), then nothing is done and the number of connections to the TAM is ***I+O***. If W is below ***I+O***, then some of the cells on the I/Os are chained.

For sequential cores (one example is given in Fig.1), if W is above or equal to {#SC×2 + 2} (#SC is the number of scan chains), then a pre-process 'Internal Chaining' will chain the internal scan chains. The value given by {#SC×2 + 2} defines the minimal number of bits needed to connect a core with scan chains using our approach (2 per scan chain, one to link the functional inputs and one to link the functional outputs). Then a 'fill' process will connect wrapper cells to internal scan chains until the total number of FFs (flip-flops) reaches the number of FFs of the longest scan chain. In Fig. 1, the scan chain of length 6 and that of length 2 are chained together. Two wrapper input cells are then connected to it, so that the total number of FFs is 10, which equals the number of FFs of the longest scan chain.

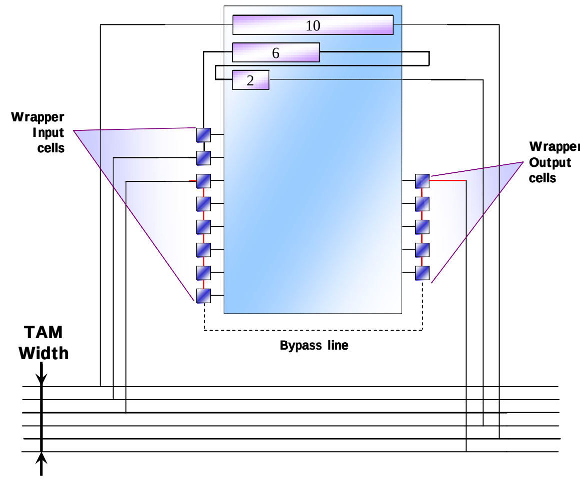

Fig. 1. Wrapper example.

If W is below {#SC×2 + 2}, the internal scan chains are chained together in order to reduce the number of needed connections to the TAM. Then the 'fill' process is applied. The simplified algorithm is presented in Fig. 2. The algorithm defines the whole curve T = f(W) which has the form of a staircase (several architectures lead to the same test time). It returns as results the Pareto-optimal points that are the ones on the left most edges at each staircase level.

We have applied the proposed algorithm to the benchmarks from ITC'02 [Benc02]. Fig. 3 shows the results for core 5 from the d695 system, which contains 32 scan chains, 38 inputs and 304 outputs with 110 vectors. We obtain the different stages on the curve corresponding to an identical test time for several W values (i.e. several architectures). For example, the Pareto-optimal point {t=10100, W=35} is optimal with respect to all points that have the same test time (t=10100) but different numbers of TAM connections, such as {t=10100, W=51}. The algorithm computes the curve for each core on a system in a very short computation time (typically a few seconds).

```
W=1000000;                    //limit for the number of TAM connections
Internal Chaining
While (W!=1)
    If (#SC == 0)             // combinational core
        If ((I+O)<=W)
            Connect one bit on every I/O wrapper cell
        Else
            Chain wrapper cells
    Else                      // sequential core
        If ((#SC×2 + 2)<=W)
            Fill procedure
            If (maxFF<I)or(maxFF<O)
                CutWrapper procedure
        Else
            Internal chaining until ((#SC×2 + 2)<=W)
            Fill procedure
            If (maxFF<I)or(maxFF<O)
                CutWrapper procedure
    Return W_needed
W=W_needed
W=W--;
End
```

Fig. 2. The wrapper design algorithm.

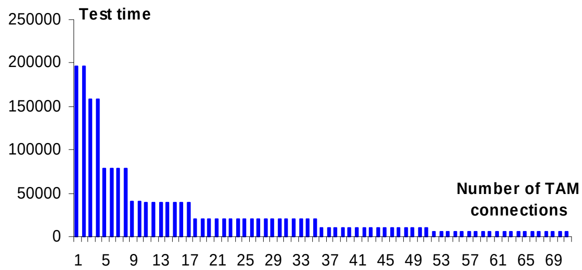

Fig. 3. Total test times of different wrapper designs for core 5 in d695.

Depending on the system test requirements, the designer can then select among the different widths and test times for each core. We have applied this algorithm to all ITC'02 benchmarks [MaIy02][Benc02], and obtained these curves for every core [Poug02].

3. TAM Architecture and Test Scheduling

Once the wrapper design is completed or a set of wrapper design alternatives is available, as provided by the wrapper design algorithm, the designer has to deal with two issues, namely test scheduling of the cores, and the design of test access architecture. The test access architecture is responsible for the transportation of the test data from the system inputs to the core inputs and from the core outputs to the system outputs. For this purpose, we make use of a generalization of the TestBus architecture.

3.1 Test Schedule

The test scheduling problem consists of two interleaved NP-complete problems (bin-packing and minimal graph-coloring) [FlPo01]. The graph-coloring and bin-packing problems cannot be approximated in bounded limits when the graph has no special structure [GaJo79]. One way to simplify the test scheduling problem is to organize tests for the target modules into so-called test sessions [Mur00][RaVe99] [ChSa97]. An alternative is to use a no session scheme, which allows minimizing the test time at the expense of area overhead and scheduling complexity. To reduce the scheduling time, a very fast heuristic based on the work by [FlPo01] has been used in the proposed approach (Fig. 4).

The complexity of this algorithm is $O(n^3)$. It can handle, for example, a schedule with more than 100 cores within one second. The algorithm works in the following way.

```
L1 = list of cores sorted by decreasing Di values (Di = test time of core i)
L2 = Ø
Tmax=0
While L1 ≠ Ø
    Place (first core in L1)
    Update Tmax
    For all others cores i in L1
        For all intervals
            If (Power, precedence, incompatibility constraints satisfied)
                If Ti+Di <= Tmax
                    Ti=Place(i)
                Else
                    remove i from the placed cores
                    L2 = L2 ∪ {i}
    L1 = L2
```

Fig. 4. The proposed scheduling algorithm.

First, the core tests are sorted in decreasing order of test time in a list, L1. While all the tests are not fully scheduled, it checks the characteristics for each core test in order, and places them as soon as possible making sure that all the constraints are satisfied. If a core cannot be scheduled due to the violation of certain constraints, it is moved to an auxiliary list L2 to be scheduled later. When the L1 list is empty, L2 moves to L1 and the process is re-iterated.

3.2 TAM Design – Pseudo Exhaustive Approach

We have first developed an approach based on a generalization of the TestBus architecture to build the TAM. The wrapper design algorithm presented in section 2 allows us to obtain different possible costs for each core. We use a hyper-graph representation for the incompatibility of the tests we have to schedule. Our approach generates all the possible incompatibility configurations (i.e. all the hyper-graphs) in order to calculate their minimal cost using a mapping heuristic ('mapping' denotes the assignment of cores to busses), and the associated schedule. Each hyper-graph corresponds to a different mapping and schedule. The algorithm (Fig.5) memorizes and finally returns the best solution in terms of test time under a given TAM constraint. It selects the Pareto-optimal points for each core (from the wrapper design algorithm), and then generates in an exhaustive way all the possible hyper-graphs (configurations) for the TAM.

The vertices of the hyper-graph correspond to the tests and are weighted by the number of connections they need on the TAM. In a similar approach in [ChSa97], the authors consider the test resources on the system and build the incompatibility graph based on them. Thus, their graph is an expression of the existing resources. In our scheme, incompatibilities, represented by the hyper-edges, can be used to capture different test conflicts and constraints. An example of hyper-graph is given in Fig. 6, where a vertex is labeled by a unique number and its weight in a parenthesis. If cores 1, 2, 3 and 4 cannot be tested pair-wisely at the same time, this information is captured by a hyper-edge connecting vertices 1, 2, 3 and 4.

We generate in theory the whole set of possible hyper-graphs for a system, and therefore all the possible architectures for the test bus. In practice, a strategy based on space pruning is used to cut the branch-and-bound search to reduce the computation time.

```
Wc = maximal width for the bus;
Ttotal = ∞;
EdgeSize=1, Depth=1;
Process (Depth, EdgeSize)
    While EdgeSize <= Number of Tests
        Generate the next edge
            Update EdgeSize
            If Pruning constraints not satisfied
                Add edge to current solution
                    If Complete solution        // i.e. containing all tests
                        Generation of incompatibility constraints;
                        T = scheduling ()
                        W = mapping ()
                        If W ≤ Wc and T ≤ Ttotal
                            Keep solution;
                    Else Process (Depth + 1, EdgeSize)
            Else Cut branch (from edge)
    End while
End
```

Fig. 5. Recursive Pseudo-Exhaustive Algorithm.

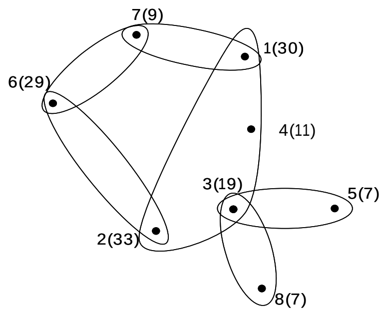

Fig. 6. Illustrative Example.

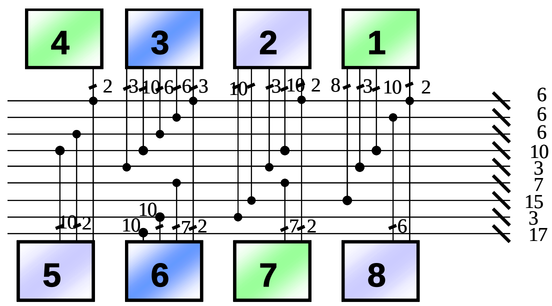

Fig. 7. Final architecture from the example in Fig. 6.

For each solution, we generate the mapping and consider the new incompatibility set to schedule the tests. The simplified algorithm is given in Fig. 5. Fig. 6 and Fig. 7 illustrate a hyper-graph example and its corresponding TAM architecture. We can also add incompatibilities (hyper-edges) to capture the conflicts due to the test of the glue logic between cores. For example, we can add a hyper-edge between cores 5 and 9, and another between core 6 and 9, with the new core 9 representing the glue logic between cores 5 and 6. Once the mapping is ready, the algorithm uses a fast heuristic to schedule the tests. The scheduling heuristic takes into account all the incompatibilities, including the added ones. We have applied the pseudo-exhaustive algorithm to the ITC'02 benchmarks.

The used width w_i and test time t_i for each core are selected manually as one of the Pareto-optimal points of the curves of each core. In Fig. 8, we show the results for the system h953 from [Benc02]. In this example, power dissipation of cores during test is taken into account and a width limit (32 bits) is fixed for the system test bus. Our scheduling algorithm takes also into account the system power limit, even though the power data are not shown in the figure. Note that the test time in this example is determined by the largest core. Nevertheless, it is possible to select other Pareto-optimal points for each core, which may be carried out manually. Checking all the possibilities for a system with ten cores with ten Pareto-optimal points for each core means checking 10^{10} possibilities.

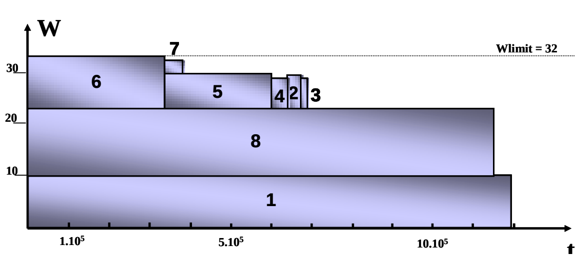

Fig. 8. Results for the h953 system.

Automatic checking for all these possibilities is therefore only efficient for systems involving a small number of cores. For larger systems, the computation time becomes too prohibitive.

3.3 TAM Design – Efficient Heuristic

We have implemented a fast heuristic in order to reduce the computation complexity. The heuristic first sorts by increasing cost order the design alternatives obtained with the wrapper design algorithm using the cost function $C_i = t_i \times w_i$. Then, it uses the scheduling approach presented in section 3.1 taking into account now also constraints on W. The algorithm, illustrated in Fig. 9 (named ScheduleW) selects appropriate wrapper designs to schedule the tests. In this algorithm, "i" is the wrapper design index. If the constraint on W is not respected, the algorithm searches the next value for (w_i t_i) with the lowest possible cost C_i. It schedules the test as soon as possible with respect to the whole set of constraints, and maximizes the TAM use.

```
....
For all intervals
    For all designs (Wi,ti)
        If (Power, precedence, incompatibility and W constraints satisfied)
            If (Ti+Di <= Tmax)
                Ti=Place(i)
            Else
                i=i+1      // try next wrapper design
    If core not scheduled
        Remove i from the placed cores
        L2 = L2 ∪{i}
....
```

Fig. 9. Algorithm ScheduleW.

4. Experimental Results and Discussions

We have applied our wrapper design, TAM design and test scheduling algorithms to the ITC'02 benchmarks. The results for system q12710, d695, p22810, and p34392 are given in Tables 2-5, respectively. For each system, results for a set of design alternatives, corresponding to different TAM widths, are reported. A row in the tables corresponds to a given TAM width. For each TAM width, the first group of columns gives the results of the pseudo-exhaustive approach, in terms of the test time indicating the quality of the test schedule, and the CPU time used to generate the solution. The last group of columns gives the corresponding results of our efficient heuristic, as well as how they compare to the other approaches.

For comparison, we have implemented another fast way to solve the TAM design and test scheduling problems by adding multiplexors to the test bus. The schedule is generated in a straightforward way: the tests are performed sequentially using the maximum TAM width available. Therefore, the Pareto-optimal points are chosen as close as possible to the TAM width limit. The results of this multiplexed approach are given in the middle column of Tables 2-5. The experimental results show clearly that our approach outperforms the multiplexed approach in terms of test scheduling lengths (on average, the test time of our approach is 21,8% smaller than that of the multiplexed approach). Note also that the multiplexed approach is not capable to handle interconnection test. When compared with the pseudo-exhaustive approach, our algorithm consumes only a tiny fraction of CPU time needed, while producing relatively comparable test scheduling results. Note also that the pseudo-exhaustive approach does not work for larger systems, such as p22810 and p34392. Therefore there are not pseudo-exhaustive results in Tables 4 and 5, nor are there any in Table 3 for TAM widths between 12 and 48 bits.

Let us look closely at one example, the system d695, which is illustrated in Fig. 10. The TAM width limit is 32 for this example and the power limitation is 1300mW. The characteristics of the cores are given in Table 1. In the ITC'02 benchmark specification, no power data are given for this system. Therefore, we add power values for each core. We have also added two precedence constraints for this system: cores 7 and 5 have to be tested before core 10, and core 6 can only be tested after cores 7 and 8 have been tested. These constraints are given in the last column of Table 1. One possible motivation for the constraints is that core 5 has to be tested first because it is a core containing potentially more faults than other cores. The generated schedule, given in Fig. 10, respects the TAM width constraint (32 bits) on the system test bus, as well as the specified precedence constraints. The dashed line in Fig. 10 represents the instantaneous dissipated power during test.

In [IyCh02] and [IyCh02a], a similar approach is presented for wrapper/TAM co-optimisation assuming that all tests are compatible. Therefore, that approach cannot deal with the cases when some tests cannot be carried out simultaneously, due to, for example, design hierarchy constraints.

Our algorithms deal, on the other hand, with the incompatible properties explicitly. Additional features of our approach are that it addresses the issues of power dissipation, precedence constraints, incompatibilities from test resource sharing, and interconnection test in a systematic manner. The issue of interconnection test is in particular important considering the increasing importance of interconnection in SoC designs. Using the wrappers, it is possible to test interconnections between two cores (which is included as a part of the wrapper design algorithm) and to add this test as a new item into the schedule (no extra wires are necessary for this test), with our approach.

Cores C_i	t_i	w_i	P_i	$Prec._i$
1	416	2	30mW	-
2	7992	3	150mW	-
3	5167	2	250mW	-
4	11129	6	100mW	-
5	10100	19	400mW	-
6	9869	19	950mW	7, 8
7	12959	10	700mW	-
8	4605	11	450mW	-
9	2820	19	350mW	-
10	7106	17	550mW	7, 5

Table 1. d695 characteristics.

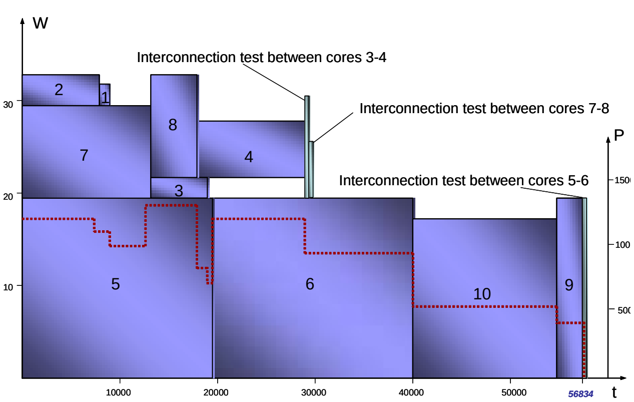

Fig. 10. The d695 schedule generated by our heuristic.

For example, on the resulting schedule of the system d695 in Fig. 10, 164 cycles are necessary to test interconnections between cores 5 and 6 and 19 wires are used (these data are extracted from the wrapper design algorithm).

5. Conclusions

We have presented several techniques developed to help the system designers to implement and optimise a test structure for the whole system. Firstly, the wrapper design is built by a fast algorithm, optimising the test time of a core for a given number of connections to the TAM. This algorithm gives all the possible implementations of the wrapper for a given core. We can then select one of the three approaches for TAM design and test scheduling. The first one uses a pseudo-exhaustive search, which entails prohibitive computation times for large systems. The other two are fast heuristics, one is based on an implementation of a multiplexed approach, and the other is a heuristic developed by us. We have demonstrated, with experiments on the ITC'02 benchmarks, the efficiency of our approach, and compared it with other approaches. Additionally, our algorithms have several interesting features, such as that it addresses precedence and power constraints and interconnection test, which are not all implemented on other approaches. In particular, the interconnection test issue is becoming more and more important for the next generation SoCs.

References

[Benc02] http://www.extra.research.philips.com/itc02socbenchm/format.html

[Chak99] K. Chakrabarty: " Test Scheduling For Core Based Systems", ICCAD'99, pp. 391-394, 1999.

[ChIy01] K. Chakrabarty, V. Iyengar, E. J. Marinissen: "Test wrapper and Test Access Mechanism Co-Optimization for System-on-Chip", ITC'01, pp. 1023-1032, 2001.

[ChSa97] 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, 1997.

[CoCa02] E. Cota, L. Caro, A. Orailoglu, M. Lubaszewski: "Test Planning and Design Space Exploration in a Core Base Environment", DATE'02, pp. 478-485, 2002.

[FlPo01] M.L. Flottes, J. Pouget, B. Rouzeyre: "Sessionless Test Scheme: Power-constrained Test Scheduling for System-on-a-Chip", VLSI-SoC'01, pp. 105-110, 2001.

[GaJo79] M.R. Garey, D. Jonhson: "Computers and Intractability: guide to the theory of NP-completeness", W.H. Freeman and Company, San Francisco, 1979.

[Harr99] P. Harrod: "Testing reusable IP – A Case Study", ITC'99, pp. 493-498, 1999.

[HuRe02] Y. Huang, S.M. Reddy, W.T. Cheng, P. Reuter, N. Mukherjee, C.C. Tsai, O. Samman, Y. Zaidan: "Optimal Core Wrapper Width Selection and SOC Test Scheduling Based on Bin Packing Algorithm", ITC'02, pp. 74-82, 2002.

[IyCh01] V.Iyengar, K. Chakrabarty: "Precedence-Based, Preemptive and Power-Constrained Test Scheduling for System-on-Chip", VLSI-SoC'01, pp. 368-374, 2001.

[IyCh02] V. Iyengar, K. Chakrabarty, E. J. Marinissen: "Efficient Wrapper/TAM Co-Optimization for Large SOCs", DATE'02, pp. 491-498, 2002.

[IyCh02a] V. Iyengar, K. Chakrabarty, E. J. Marinissen: "On Using Rectangle Packing for SoC Wrapper/TAM co-optimization", VTS'02, pp. 253-258, 2002.

[MaGo00] E. J. Marinissen, S. K. Goel and M. Lousberg: "Wrapper Design for Embedded Core Test", ITC'00, pp. 911-920, 2000.

[MaIy02] E. J. Marinissen, V. Iyengar, K. Chakrabarty: "A set of Benchmarks for Modular Testing of SoCs", ITC'02, pp. 519-528, 2002.

[Mur00] V. Muresan, X. Wang, M. Vladutiu, V. Muresan: " A comparison of classical Scheduling Approaches in Power-Constrained Block-Test Scheduling ", ITC'00, pp. 882-891, 2000.

[NiAl99] Nicola Nicolici, B.M. Al-Hashimi: "Power Conscious Test Synthesis and Scheduling for BIST RTL Data Paths", ATS'99, pp. 107-112.

[NoPa01] M. Nourani and C. Papachristou: "An ILP Formulation to optimize Test Access Mechanism in System-on-Chip Testing", ITC'00, pp. 902-910, 2000.

[Poug02] Julien Pouget: "Embedded Test for System-On-Chips: Test Scheduling and Architectural Solutions", PhD Thesis, 2002. www.ida.liu.se/~g-julpo/

[RaVe99] C.P. Ravikumar, A. Verma, G. Chandra: "A Polynomial-Time Algorithm for Power Constrained Testing of Core Based Systems", ATS'99, pp. 107-112.

[Zori98] Y. Zorian: "System-On-Chip Test Strategies", DAC'98, pp. 752-756.

q12710	1 - Exhaustive TAM approach		2 - Multiplexed approach		3 - Our Heuristic		Comparison			
TAM Width	Test time	Cpu time	Test time	Cpu time	Test time	Cpu time	Test time		Cpu time	
							(3) vs (1)	(3) vs (2)	(3) vs (1)	(3) vs (2)
32	2 644 464	0,01s	6 228 966	0,001s	2 644464	0,001s	0 %	- 57,5 %	- 90 %	0 %
24	3 096 765	0,01s	6 228 966	0,001s	3 177502	0,001s	+ 2,6 %	- 49 %	- 90 %	0 %
20	3 177 502	0,5s	6 228 966	0,001s	3 177502	0,001s	0 %	- 49 %	- 99,8 %	0 %
16	4 368 020	0,9s	6 228 966	0,001s	5 146524	0,001s	+ 17,8 %	- 17,4 %	- 99,9 %	0 %
12	5 146 524	0,7s	6 651 981	0,001s	5 146524	0,001s	0 %	- 22,6 %	- 99,8 %	0 %
10	6 377 663	0,8s	6 651 981	0,001s	6 377663	0,001s	0 %	- 4,1 %	- 99,9 %	0 %

Table 2. Results on q12710 benchmark.

d695	1 - Exhaustive TAM approach		2 - Multiplexed approach		3 - Our Heuristic		Comparison			
TAM Width	Test time	Cpu time	Test time	Cpu time	Test time	Cpu time	Test time		Cpu time	
							(3) vs (1)	(3) vs (2)	(3) vs (1)	(3) vs (2)
80	20090	6min	36 232	0,015s	25648	0,0156s	+ 27,7 %	- 29,2 %	- 100 %	+ 4 %
64	28369	>180min	45 798	0,015s	32 857	0,0156s	+ 15,8 %	- 28,1 %	- 100 %	+ 4 %
48	-	-	45 972	0,015s	33 031	0,0156s	-	- 28,1 %	-	+ 4 %
32	-	-	78 077	0,015s	65 136	0,0156s	-	- 16,6 %	-	+ 4 %
24	-	-	78 386	0,015s	71 274	0,0156s	-	- 9,1 %	-	+ 4 %
20	-	-	78 547	0,015s	76 040	0,0156s	-	- 3,2 %	-	+ 4 %
16	-	-	142 683	0,015s	122198	0,0156s	-	- 14,4 %	-	+ 4 %
12	-	-	143 153	0,015s	143 153	0,0156s	-	- 0 %	-	+ 4 %

Table 3. Results on d695 benchmark.

p22810	2 - Multiplexed approach		3 - Our Heuristic		Comparison	
TAM Width	Test time	Cpu time	Test time	Cpu time	Test time	Cpu time
					(3) vs (2)	(3) vs (2)
80	503 635	0,08s	223 463	0,172s	- 55,6 %	+ 115 %
64	531 631	0,069s	294 046	0,078s	- 44,7 %	+ 13 %
48	619 537	0,063s	418 226	0,068s	- 32,5 %	+ 7,9 %
32	664 665	0,063s	574 120	0,069s	- 13,6 %	+ 9,5 %
24	848 601	0,062s	805 741	0,07s	- 5,1 %	+ 12,9 %
20	1 013 766	0,062s	973 632	0,069s	- 4,0 %	+ 11,3 %
16	1 094 717	0,061s	1 070 826	0,069s	- 2,2 %	+ 13,1 %
12	1 588 681	0,061s	1 507525	0,069s	- 5,1 %	+ 13,1 %

Table 4. Results on p22810 benchmark.

p34392	2 - Multiplexed approach		3 - Our Heuristic		Comparison	
TAM Width	Test time	Cpu time	Test time	Cpu time	Test time	Cpu time
					(3) vs (2)	(3) vs (2)
80	1 389 219	0,55s	561 139	0,56s	- 59,6 %	+ 1,8 %
64	1 389 677	0,55s	652 118	0,59s	- 53,1 %	+ 7,2 %
48	1 422 148	0,53s	990 009	0,56s	- 30,4 %	+ 5,6 %
32	1 749 945	0,53s	1 575 101	0,44s	- 10,0 %	- 17 %
24	1 805 535	0,55s	1 635 259	0,48s	- 9,4 %	- 12,7 %
20	1 917 813	0,53s	1 905 563	0,47s	- 0,6 %	- 11,3 %
16	3 288 694	0,53s	3 118 418	0,45s	- 5,1 %	- 15,1 %
12	3 390 156	0,53s	3 376 448	0,42s	- 0,7 %	- 17 %

Table 5. Results on p34392 benchmark.