An Integrated Framework for the Design and Optimization of SOC Test Solutions

Erik Larsson and Zebo Peng

Embedded Systems Laboratory
Department of Computer and Information Science,
Linköpings Universitet, Sweden.

Abstract[1]

We propose an integrated framework for the design of SOC test solutions, which includes a set of algorithms for early design space exploration as well as extensive optimization for the final solution. The framework deals with test scheduling, test access mechanism design, test sets selection, and test resource placement. Our approach minimizes the test application time and the cost of the test access mechanism while considering constraints on tests and power consumption. The main feature of our approach is that it provides an integrated design environment to treat several different tasks at the same time, which were traditionally dealt with as separate problems. We have made an implementation of the proposed heuristic used for the early design space exploration and an implementation based on Simulated Annealing for the extensive optimization. Experiments on several benchmarks and industrial designs show the usefulness and efficiency of our approach.

1. Introduction

The testing of System-on-Chip (SOC) is a crucial and time consuming problem due to the increasing design complexity. Therefore it is important to provide the test designer with support to develop an efficient test solution.

The work-flow for a test designer developing a test solution consists typically of two consecutive parts: an early design space exploration and an extensive optimization for the final solution. During the process, conflicts and limitations must be carefully considered. For instance, tests may be in conflict with each other due to the sharing of test resources; and power consumption must be controlled, otherwise the system may be damaged during test. Furthermore, test resources such as external testers support a limited number of scan-chains and have a limited test memory, which also introduce constraints.

Research has been going on in developing techniques for test scheduling, test access mechanism (TAM) design and testability analysis. For example, a framework to support the design of test solutions for SOC with Built-In Self-Test (BIST) has been proposed by Benso *et al.* [2]. In this paper, we combine and generalize several approaches in order to create an integrated framework for the development of SOC test solutions where:

- tests are scheduled to minimize the total test time,
- a minimal TAM is designed,
- test resources are floor-planned, and
- test sets for each core with test resources are selected.

The above set of tasks is performed in a single algorithm, which considers test conflicts, power limitation and test resource constraints [17,18,19]. The algorithm is suitable for early design space exploration due to its low computational complexity, which is an advantage since it will be used iteratively many times.

Chakrabarty showed that test scheduling is equal to the open-shop scheduling [4], which is known to be NP-complete and the use of heuristics are therefore justified. Several heuristics have been proposed [1,4,5,9,11,14,23, 27]; however, they have been evaluated using rather small benchmarks. For such benchmarks, a technique based on Mixed-Integer Linear-Programming (MILP) can be used [4]. A disadvantage is the complexity of solving the MILP model since the size of it quickly grows with the number of tests making it infeasible for large industrial designs. We have therefore made an implementation based on Simulated Annealing for the extensive optimization of the test schedule and the TAM design for the final solution [16]. We have performed experiments on several benchmarks and on an Ericsson design consisting of 170 tests to show the efficiency and usefulness of our approach.

The rest of the paper is organised as follows. Related work is outlined in Section 2 and the system modelling is introduced in Section 3. Factors affecting the test solution are presented in Section 4. The integrated algorithm is described in Section 5 and how we used Simulated Annealing is outlined in Section 6. The paper is concluded with experimental results in Section 7 and conclusions are in Section 8.

1. This work has partially been supported by the Swedish Agency for Innovation Systems (VINNOVA) and Ericsson.

2. Related Work

The basic problem in test scheduling is to assign a start time for all tests. In order to minimize the test application time, tests are scheduled as concurrent as possible, however, various types of constraints must be considered.

A test to be scheduled consists of a set of test vectors produced or stored at a test source (placed on-chip or off-chip). The test response from the test is evaluated at a test sink (placed on-chip or off-chip). When applying a test, a test conflict may occur, which must be considered during the scheduling process. For instance, often a testable unit is tested by several test sets (usually an external test set and a on-chip test set are required to reach high test quality). If several tests are used for a testable unit, only one test can be applied to the testable unit at a time.

Constraints on power consumption must be considered otherwise the system can be damaged. During testing mode, the power consumption is usually higher compared to normal operation [12]. For instance, consider a memory, which often is organized in memory banks. During normal operation, only a single bank is activated. However, during testing mode, in order to test the system in the shortest possible time it is desirable to concurrently activate as many banks as possible [9].

The use of different test resources may entail constraints on test scheduling. For instance, external testers have limitations of bandwidth due to that a scan chain operates usually at a maximum frequency of 50 MHz [13]. External testers can usually only support a maximum of 8 scan chains [13], resulting in long test application time for large designs. Furthermore, an external tester's memory is limited by its size [13].

Zorian proposed a test scheduling technique for fully BISTed systems where test time is minimized while power constraint is considered [27]. The tests are scheduled in sessions where tests at cores placed physically close to each other are grouped in the same test session. In a fully BISTed system, each cores has its dedicated test source and test sink; and there might not be any conflicts among tests, *i.e.* the tests can be scheduled concurrently. However, in the general case, conflicts among tests may occur. Garg *et al.* proposed a test scheduling technique where test time is minimized for systems with test conflicts [11] and for core-based systems a test scheduling technique is proposed by Chakrabarty [4, 5]. Chou *et al*. proposed an analytic test scheduling technique where test conflicts and power constraints are considered [9]. A resource graph is used to model the system where an arc between a test and a resource indicate that the resource is required for the test, Figure 1. From the resource graph, a test compatibility graph (TCG) is generated (Figure 2) where each test is a node and an arc between two nodes indicates that the tests can be scheduled concurrently. For instance t_1 and t_4 can be scheduled at the same time. Each test is attached with its test time and its power consumption and the maximal allowed power consumption is 10. The tests t_1 and t_5 are compatible, however, due to the power limit they cannot be scheduled at the same time.

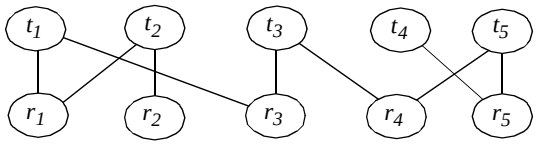

Figure 1. Resource graph of an example system.

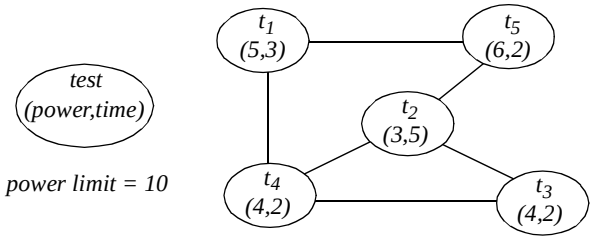

Figure 2. Test compatibility graph (TCG) of the example system (Figure 1).

Another test scheduling approach is proposed by Muresan *et al*. where constraints among tests and power consumption are considered [23]. Favour is given to reduce the test time by allowing new tests to start even if all tests in a session are not completed. Iyengar and Chakrabarty proposed a pre-emptive test scheduling technique where the test for a testable unit may be interrupted and resumed later, *i.e*. the test set is partitioned into several test sets [14]. Using a scheme by Craig *et al.*, the above discussed scheduling techniques can be grouped in [10]:

- nonpartitioned testing,
- partitioned testing with run to completion, and
- partitioned testing.

The differences among the grouping are illustrated with five tests (t_1,..., t_5) in Figure 3. The scheduling strategies proposed by Zorian [27] and Chou *et al.* [9] are nonpartitioned (Figure 3(a)), the strategy proposed by Muresan *et al.* is partitioned testing with run to completion (Figure 3(b)) and the approach proposed by Iyengar and Chakrabarty [14] is partitioned testing (Figure 3(c)).

A test infrastructure is responsible for the transportation of test vectors from test sources to cores under test and test responses from cores under test to test sinks. It consists of two parts; one for the test data transportation and one for the control of the transportation. In the approach for fully BISTed systems proposed by Zorian [27], tests at cores placed physically close to each other are grouped in the same test session. The main advantage is that the same control structure can be used for all tests in the session, which minimizes the routing of control wires. In general,

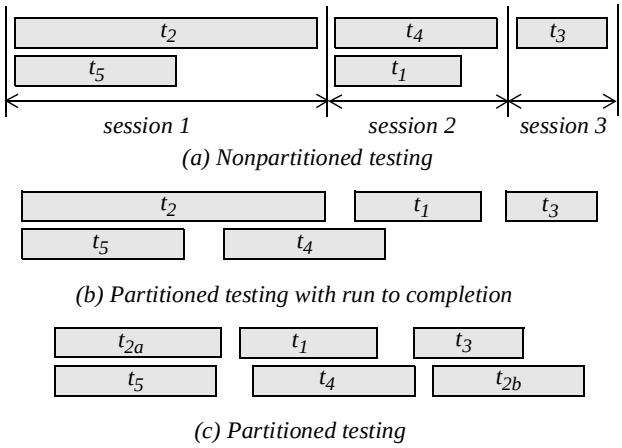

Figure 3. Scheduling approaches.

systems are not tested with a BIST structure only for each testable unit and therefore a TAM is required [6,7,8,24]. Chakrabarty proposed an integer linear programming (ILP) for the allocation of TAM width [6,7,8] and the effect on test time for systems using various design styles for test access with the TestShell wrapper is analysed by Aertes and Marinissen [1].

To ease test access the cores can be placed in wrappers such as Boundary scan [3], TestShell [20] or P1500 [15, 21]. The Boundary scan technique, developed for PCB designs, suffers from long testing time due to the shifting process, and it becomes even worse for SOC designs since the amount of test data to be transported increases. An alternative is to use an architectures as proposed by Aertes and Marinissen (Figure 4.(a,b,c)) where the test time only depends on the number of flip-flops within the cores and the number of test vectors, *i.e.* the transportation to and from a core is free [1]. The daisy-chained architecture uses a clocked bypass, however, Marinissen *et al.* have also proposed a wrapper design library allowing bypass structures without clock delay [22].

The above scheduling techniques all assume that all testable units have their selected test sets. Sugihara *et al.* proposed a technique for selecting test sets where each core may be tested by one test set from an external tester and one test set from a dedicated test generator for the core [25].

3. System modelling

This section describes the formal notation we use to model the SOC under test. An example of a system under test is given in Figure 5 where each core is placed in a wrapper to ease test access. A DFT technique is added to each core and in this example all cores are tested using the scan technique. The test access port (*tap*) is the connection to an external tester and for instance the test source, *test source* 1, and test sink, *test sink* 1, are implemented on-chip [17,19]. Applying several sets of tests, where each test set is produced or

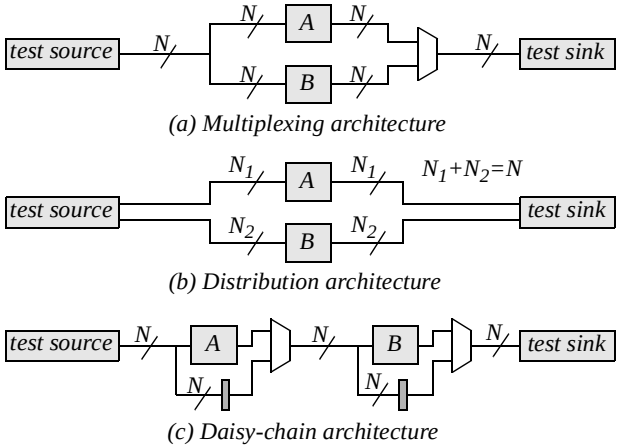

Figure 4. Multiplexing, distribution and daisy-chain architecture [1].

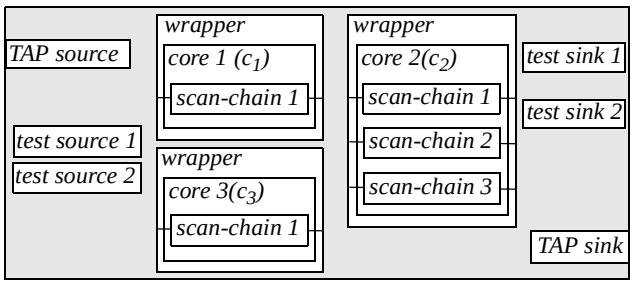

Figure 5. An example system.

stored at a test source and the test response is analysed at a test sink, tests the system.

The system in Figure 5 can be modelled as a *design with test*, $DT = (C, R_{source}, R_{sink}, p_{max}, T, source, sink, core, constraint, mem, bw)$, where:

$C = \{c_1, c_2,..., c_n\}$ is a finite set of cores where each core $c_i \in C$ is characterized by $p_{idle}(c_i)$: idle power;

$R_{source} = \{r_1, r_2,..., r_p\}$ is a finite set of test sources;

$R_{sink} = \{r_1, r_2,..., r_q\}$ is a finite set of test sinks;

p_{max}: maximal allowed power at any time;

$T = \{t_1, t_2,..., t_o\}$ is a finite set of tests, each consisting of a set of test vectors. Several tests form a *core test* (*CT*) and each core, c_i, can be associated with several core tests, CT_{ij} (j=1,2,...,l).

Each test t_i is characterized by:

$\tau_{test}(t_i)$: test time for test t_i,

$p_{test}(t_i)$: test power for test t_i,

$bw(t_i)$: bandwidth required for t_i

$mem(t_i)$: memory required for test pattern storage.

source: $T \rightarrow R_{source}$ defines the test sources for the tests;

sink: $T \rightarrow R_{sink}$ defines the test sinks for the tests;

core: $T \rightarrow C$: the core where a test is applied;

constraint: $T \rightarrow 2^C$: the set of cores required for a test;

$mem(r_i)$: memory available at test source $r_i \in R_{source}$;

$bw(r_i)$: bandwidth at test source $r_i \in R_{source}$.

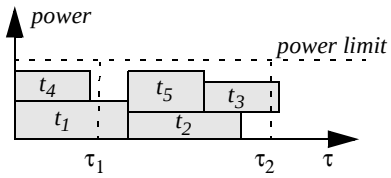

Figure 6. Example of test scheduling.

For each test, one test sink and one test source are required. In our model, it is possible for every test to combine any type of test source (on-chip or off-chip) with any type of test sink (on-chip or off-chip), which is important to allow high flexibility. To further give the designer a high flexibility to explore different combinations of tests, it is possible to define several set of tests (CT_{ij}) for each core where each such set tests the core.

Given a system as in Figure 5 where for each of the cores c_1, c_2, c_3, one CT exists. For core c_1, $CT_{11}=\{t_1, t_2\}$, for core c_2, $CT_{21}=\{t_4, t_5\}$ and for c_3, $CT_{31}=\{t_3\}$. The on-chip test source 1 is required by t_1 and t_3 and the on-chip test sink 1 is required by t_3 and t_5. The resource graph in Figure 1 and the test compatibility graph in Figure 2 model the system when r_3 (test source 1) is needed by t_1 and t_3 and r_4 (test sink 1) needed by t_3 and t_5.

4. The SOC Test Issues

In this section we discuss the different issues considered by our SOC test framework. We also demonstrate the importance of considering them in an integrated manner.

4.1 Test Scheduling

The test scheduling problem can be seen as placing all tests in a diagram as in Figure 6 while satisfying all constrains. For instance, t_1 and t_2 cannot be applied at the same time since both tests the same core, c_1 (example in Section 3).

The basic difference between our scheduling technique [17,19] and the approaches proposed by Zorian [27] and Chou *et al.* [9] is, besides that we design the TAM while scheduling the tests, that Zorian and Chou *et al.* do not allow new tests to start until all tests in a session are completed. It means that if t_3 is about to be scheduled it cannot be scheduled as in Figure 6. In the approach proposed by Muresan *et al.* [23], t_3 can be scheduled if it is completed no later than t_2.

In our approach it is optional if tests may start before all tests in a session are completed (nonpartitioned testing) or not (partitioned testing with run to completion). The advantage of using the latter is that it gives more flexibility.

Let a schedule S be an ordered set of tests such that:

$$\{S(t_i) < S(t_j) \,|\, \tau_{start}(t_i) \le \tau_{start}(t_j),\ i \ne j,\ \forall t_i \in S,\ \forall t_j \in S\},$$

where $S(t_i)$ defines the position of test t_i in S; $\tau_{start}(t_i)$ denotes the time when test t_i is scheduled to start, and $\tau_{end}(t_i)$ its completion time: $\tau_{end}(t_i) = \tau_{start}(t_i) + \tau_{test}(t_i)$.

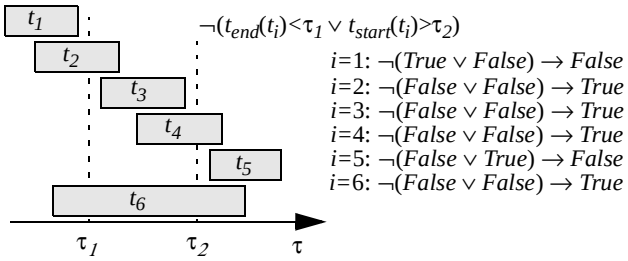

Figure 7. The function *scheduled*.

For each test, t_i, the start time and the wires for test data transportation have to be determined before it is inserted into the schedule, S.

Let the Boolean function *scheduled*(t_i, τ_1, τ_2) be true if test t_i is scheduled in such a way that the test time overlaps with the time interval [τ_1, τ_2], *i.e.*,

$$\{t_i \in S \wedge \neg(\tau_{end}(t_i) < \tau_1 \vee \tau_{start}(t_i) > \tau_2)\}.$$

An example to illustrate the function *scheduled* for a set of scheduled tests is shown in Figure 7.

The Boolean function *scheduled*(r_i, τ_1, τ_2) is true if a source r_i is used by a test t_j between τ_1 and τ_2, *i.e.*:

$$\{\exists t_j \in S \,|\, r_i = source(t_j) \wedge scheduled(t_j, \tau_1 \tau_2)\}.$$

A similar definition is used if a sink r_i is scheduled (used by any test) between τ_1 and τ_2. The Boolean function *scheduled*(*constraint*(t_i), τ_1, τ_2) is true if:

$$\{\exists t_j \in S \,|\, core(t_j) \in constraint(t_i) \wedge scheduled(t_j, \tau_1, \tau_2)\}.$$

4.2 Power Consumption

Generally speaking, there are more switching activities during the testing mode of a system than when it is operated under the normal mode. The power consumption of a CMOS circuit is given by a static part and a dynamic part where the latter dominates and can be characterized by:

$$p = C \times V^2 \times f \times \alpha$$

where the capacitance C, the voltage V, and the clock frequency f are fixed for a given design [26]. The switch activity α, on the other hand, depends on the input to the system, which during testing are test vectors and therefore the power dissipation vary depending on the test vectors.

An example illustrating the test power dissipation variation over time τ for two test t_i and t_j is in Figure 8. Let $p_i(\tau)$ and $p_j(\tau)$ be the instantaneous power dissipation of two compatible tests t_i and t_j, respectively, and $P(t_i)$ and $P(t_j)$ be the corresponding maximal power dissipation.

If $p_i(\tau) + p_j(\tau) < P_{max}$, the two tests can be scheduled at the same time. However, instantaneous power of each test vector is hard to obtain. To simplify the analysis, a fixed value $p_{test}(t_i)$ is usually assigned for all test vectors in a test t_i such that when the test is performed the power dissipation is no more then $p_{test}(t_i)$ at any moment.

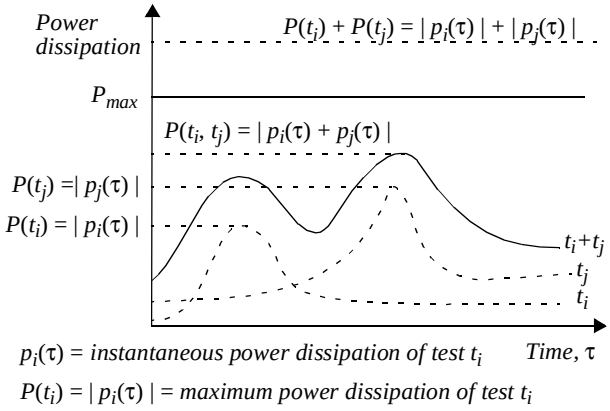

Figure 8. Power dissipation as a function of time [9].

The $p_{test}(t_i)$ can be assigned as the average power dissipation over all test vectors in t_i or as the maximum power dissipation over all test vectors in t_i. The former approach could be too optimistic, leading to an undesirable test schedule, which exceeds the test power constraints. The latter could be too pessimistic; however, it guarantees that the power dissipation will satisfy the constraints. Usually, in a test environment the difference between the average and the maximal power dissipation for each test is often small since the objective is to maximize the circuit activity so that it can be tested in the shortest possible time [9]. Therefore, the definition of power dissipation $p_{test}(t_i)$ for a test t_i is usually assigned to the maximal test power dissipation ($P(t_i)$) when test t_i alone is applied to the device. This simplification was introduced by Chou *et al.* [9] and has been used by Zorian [27] and by Muresan *et al.* [23].We will use this assumption also in our approach.

Let $p_{sch}(\tau_1, \tau_2)$ denote the peak power between τ_1 to τ_2:

$$max\left\{ \sum_{\forall t_i scheduled(t_i, \tau)} p_{test}(t_i) - p_{idle}(core(t_i)) + \sum_{\forall c_i \in C} p_{idle}(c_i), \tau \in [\tau_1, \tau_2] \right\},$$

where $scheduled(t_i, \tau)$=$scheduled(t_i, \tau, \tau)$.

As an example, applying the function $p_{sch}(\tau_1, \tau_2)$ on the schedule for a system with 5 tests as in Figure 6, with τ_1 and τ_2 as indicated in the figure, returns $p_{test}(t_2) + p_{test}(t_5) + p_{idle}(c_3)$, the peak power consumption between τ_1 and τ_2.

In our approach, the maximal power consumption should not exceed the power constraint, p_{max}, for a schedule to be accepted. That is, $p_{sch}(0, \infty) \leq p_{max}$.

4.3 Test Source Limitations

A test source usually has a limited bandwidth. For instance, external tester may only support a limited number of scan chains at a time [13]. There could also be a limitation in number of available pins. For each test source, this information is given in the attribute bandwidth.

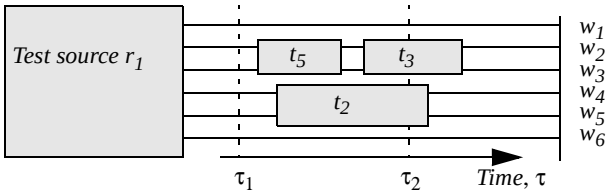

Figure 9. The TAM allocation.

The function $bw_{alloc}(r_i, \tau_1, \tau_2)$ gives the maximal number of wires allocated between τ_1 and τ_2 for a source r_i, *i.e.*:

$$max\left\{ \sum_{\forall t_j scheduled(t_j, \tau) \wedge r_i = source(t_j)} \left| t_{wires}(t_j) \right|, \tau \in [\tau_1, \tau_2] \right\}.$$

For instance, in Figure 9 a test source r_1 feeds test t_5 using wire w_2 and w_3. In this example, $bw(r_1)$=6 and $bw_{alloc}(r_i, \tau_1, \tau_2)$ = 4 since wire w_2 and w_3 are used by t_3 and t_5 and w_5 and w_6 are used by t_2.

A test generator (test source) may use a memory for storing the test patterns. In particular, external test generators use such a memory with a limited size, which may lead to additional constraints on test scheduling [13].

The function $mem_{alloc}(r_i, \tau_1, \tau_2)$ gives the allocated memory between time τ_1 and τ_2 for a given source r_i, *i.e.*:

$$max\left\{ \sum_{\forall t_j scheduled(t_j, \tau) \wedge r_i = source(t_j)} mem(t_j), \tau \in [\tau_1, \tau_2] \right\}.$$

4.4 Test Set Selection

A test set is attached with its test power consumption, test application time, bandwidth requirement and memory requirement (Section 3). The required test source and the test sink are also defined for each test set. In the approach proposed by Sugihara *et al.* each testable unit can be tested by two test sets using a dedicated BIST resource and an external tester [25]. In our approach, we assume that an arbitrary number of test sets can be used for each testable unit where each test set can be specified using any type of test resource. For instance, we allow the same test resource to be used by several test sets at different testable units. The major advantage is that the designer can define and explore a wider range of test sets. Due to that the test resources are defined for each test set it is possible to make a comparison of different test sets not only in terms of the number of test vectors but also in respect to test resources.

For each testable unit, the designer defines a core test set where each set is a set of test vectors. For instance, if the following CT_{11}={t_1, t_2}, CT_{12}={t_6}, CT_{13}={t_7, t_8, t_9} is given for core c_1. One of CT_{11}, CT_{12} or CT_{13} must be selected and all tests with in the selected CT must be scheduled and no other tests from any other CT for core c_1 should be scheduled.

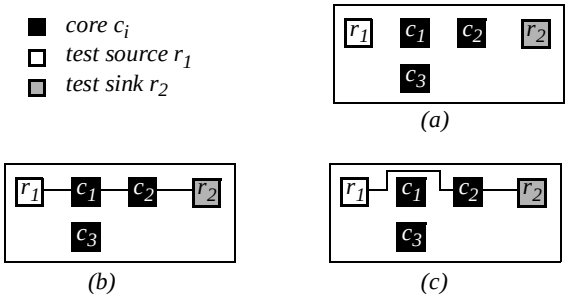

Figure 10. Illustrating TAM design alternatives using the example in Figure 5.

4.5 Test Access Mechanism

A test infrastructure transports and controls the flow of test data in the system under test. The Boundary scan technique can be used for these purposes, however, it suffers from long testing times and due to the amount of test data to be transported in the system, we assume an infrastructure consisting of test wires (lines). It means we add needed number of test wires from test sources to cores and from cores to test sinks. The time to transport a test vector from a test source to a core and the time to transport test response from a core to a test sink is neglected, which means that the test time is determined by the test vectors and design characteristics of each core.

When adding a TAM between a test source and a core or between a core and a test sink, and the test data has to pass another core, c_i, several routing options are possible:

1. through the core c_i using its transparent mode;
2. through an optional bypass structure of core c_i; and
3. around core c_i and not connecting it to the TAM.

The model in Figure 10(a) of the example system in Figure 5 illustrates the advantage of alternatives 1 and 2 (Figure 10 (b)) compared to alternative 3 (Figure 10 (c)) since the TAM can be reused in the former two cases. However, a delay may be introduced when the core is in transparent mode or its by-pass structure is used as in the TestShell [21]. On the other hand, Marinissen *et al.* recently proposed a library of wrapper cells allowing a flexible design where it is possible to design non-clocked bypass structures of TAM width [22]. In the following, we assume that bypass may be solved using such non-delay mechanism.

When designing the TAM, two problems must be solved:

- the design and routing of the TAMs, and
- the scheduling of tests using the TAMs.

In order to minimize the routing, few and short wires are desired. However, such approach increases the test time of the system. For instance, consider System S [5] (Table 1) where we added the floor planning (x, y co-ordinates for each core). A minimal TAM would be a single wire starting at the TAP, connecting all cores and ending at the TAP.

Core	Index i	External test cycles, e_i	BIST cycles, b_i	Placement	
				x	y
c880	1	377	4096	10	10
c2670	2	15958	64000	20	10
c7552	3	8448	64000	10	30
s953	4	28959	217140	20	30
s5378	5	60698	389214	30	30
s1196	6	778	135200	30	10

Table 1. Test data for the cores in System S.

However, such TAM leads to high test application time since no tests can be applied concurrently, all tests have to be applied in a sequence.

The system (for instance the example system in Figure 5 or System S) can be modelled as a directed graph, $G=(V,A)$, where V consists of the set of cores (the testable units), C, the set of test sources, R_{source}, and the set of test sinks, R_{sink}, *i.e.* $V=C\cup R_{source}\cup R_{sink}$ [17,19]. An arc $a_i\in A$ between two vertices v_i and v_j indicates a TAM (a wire) where it is possible to transport test data from v_i to v_j. Initially no TAM exists in the system, *i.e.* $A=\emptyset$. However, if the functional infrastructure may be used, it can be included in A initially. A test wire w_i is a path of edges $\{(v_0,v_1),..,(v_{n-1},v_n)\}$ where $v_0\in R_{source}$ and $v_n\in R_{sink}$. Let Δy_{ij} be defined as $|y(v_i)-y(v_j)|$ and Δx_{ij} as $|x(v_i)-x(v_j)|$, where $x(v_i)$ and $y(v_i)$ are the *x-placement* respectively the *y-placement* for a vertex v_i and the distance between vertex v_i and v_j is:

$$dist(v_i, v_j) = \sqrt{(\Delta y_{ij})^2 + (\Delta x_{ij})^2}.$$

The Boolean function $scheduled(w_i, \tau_1, \tau_2)$ is true when a wire w_i is used between τ_1 to τ_2:

$$\{\exists t_j \in S | w_i \in tam(t_j) \wedge scheduled(t_j, \tau_1, \tau_2)\},$$

where $tam(t_j)$ is the set of wires allocated for test t_j.

The information of the nearest core in four direction, *north*, *east*, *south* and *west*, are stored for each vertex. The function $south(v_i)$ gives the closest vertex south of v:

$$south(v_i) = \left\{\left(\frac{\Delta y_{ij}}{\Delta x_{ij}} > 1 \vee \frac{\Delta y_{ij}}{\Delta x_{ij}} < -1\right),\right.$$

$$y(v_j) < y(v_i),\ i \neq j,\ min\{dist(v_i, v_j)\}.$$

The functions $north(v_i)$, $east(v_i)$ and $west(v_i)$ are defined in similar ways. The function $insert(v_i, v_j)$ inserts a directed arc from vertex v_i to vertex v_j *if and only if* the is true:

$$\{south(v_i, v_j) \vee north(v_i, v_j) \vee west(v_i, v_j) \vee east(v_i, v_j)\}.$$

The function $closest(v_i, v_j)$ gives a vertex, v_k, in the neighbourhood of v_i with the shortest distance to v_j. The function $add(v_i, v_j)$ adds arcs from v_i to v_j in the following way: (1) find $v_k=closest(v_i, v_j)$; (2) add an arc from v_i to v_k; (3) if $v_k = v_j$, terminate otherwise let $v_i=v_k$ and go to (1).

The total length of a path is the sum of all individual edges. An example to illustrate the calculation of the length

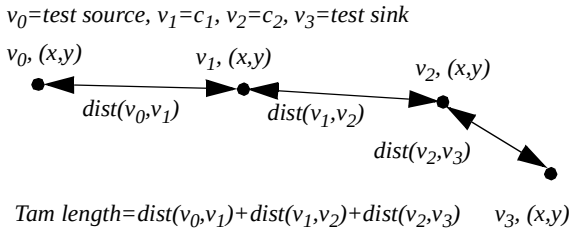

Figure 11. Computing the TAM length.

of a test wire on the example system (Figure 5) defined as a path is in Figure 11 (in this path core c_3 is not included).

4.6 Test Floor-planning

In the approach proposed by Sugihara *et al.* each testable unit is tested by a test sets using a dedicated BIST resource and an external tester [25]. However, we allow the sharing of test resources and if a BIST resource is shared among several cores, the floor-planning is not trivial.

For instance, if two cores in the example design (Figure 5) both uses a single on-chip test source and test sink, it is most feasible to place the test source at one core while the test sink is placed at the other core.

Initially, the test resources may not be placed. Our algorithm described in the next section, determines their placement in this case.

5. The Heuristic Algorithm

In this section the issues discussed above are combined into an algorithm. The algorithm assumes that the tests are initially sorted according to a key *k*, which characterizes *power*(*p*), *test time*(*t*) or *power*×*test time*(*p*×*t*).

Let *T* be the ordered set of the tests, which are ordered based on the key *k*. If the scheduling approach is partitioned testing with run to completion the function *nexttime*(t_{old}) gives the next time where it is possible to schedule a test:

$$\{\tau_{end}(t_i) \mid min(\tau_{end}(t_i)),\ \tau_{old} < \tau_{end}(t_i),\ \forall t_i \in S\},$$

otherwise if nonpartitioned testing is used the function *nexttime*(t_{old}) is defined as:

$$\{\tau_{end}(t_i) \mid max(\tau_{end}(t_i)),\ \tau_{old} < \tau_{end}(t_i),\ \forall t_i \in S\}.$$

The algorithm depicted in Figure 12 can basically be divided into four parts for:

- constraint checking,
- test resource placement,
- test access mechanism design and routing, and
- test scheduling.

A main loop is terminated when there exists a core test (CT) for each core such that all tests within the selected CT are scheduled. In each iteration at the loop over the tests in *T* a test *cur* is checked. A check is also made to determine if all

```
Sort T according to the key (p, t or p×t);
S=∅; τ=0;
until ∀b_pq ∃CT_pq ∀t_s∈S do
  for all cur in T do
    v_a=source(cur);
    v_b=core(cur);
    v_c=sink(cur);
    τ_end=τ+t_test(cur);
    if all constraints are satisfied then begin
      ¬scheduled(v_a, 0, t_end) floor-plan v_a at v_b;
      ¬scheduled(v_c, 0, t_end) floor-plan v_c at v_b;
      for all required test resources begin
        new=length of a new wire w_j connecting v_a, v_b and v_c;
        u=number of wires connecting v_a, v_b and v_c not
          scheduled from τ to τ_end;
        v=number of wires connecting v_a, v_b and v_c;
        for all min(v-u, bw(cur))wires w_j
          extend=extend+length of an available wire(w_j);
        if (bw(cur)>u)
          extend=extend+new×(par-u);
          move=par(v_a) × min{dist(v_a, v_b),dist(v_b, v_c)};
          if (move≤min{extend, new × bw(cur)})
            v_x, v_y=min{dist(v_a, v_b), dist(v_b, v_c)}, dist(v_a, v_b)>0,
                dist(v_b,v_c)>0
            add par(v_a) wires between v_x and v_y;
            if (v_x=source(cur)) then floorplan v_a at v_b;
            if (v_y = sink(cur)) then floorplan v_c at v_b;
      end
      for r = 1 to bw(cur)
        if there exists a not scheduled wire during τ to τ_end
          connecting v_a, v_b and v_c it is selected
        else
          if (length of a new wire < length of extending a wire w_j)
            w_j=add(v_a, v_b)+add(v_b, v_c);
          else
            extend wire;
      schedule cur and remove cur from T;
    end;
  τ = nexttime(τ).
```

Figure 12. The system test algorithm.

constraints are fulfilled, *i.e.* it is possible to schedule test *cur* with a start at τ and an end time at $\tau_{end}=\tau+\tau_{test}$:

- $\neg\exists t_f\ (t_f \in CT_{ij} \wedge t_f \in S \wedge cur \notin CT_{ij})$ checks that another core test set for current core is not used in the schedule,
- $p_{sch}(\tau, \tau_{end})+p_{test}(cur)<p_{max}$ checks that the power constraint is not violated,
- $\neg scheduled(v_a, \tau, \tau_{end})$ checks that the test source is not scheduled during τ to τ_{end},
- $\neg scheduled(v_c, \tau, \tau_{end})$ checks that the test sink is not scheduled during τ to τ_{end},
- $\neg scheduled(constraint(cur), \tau, \tau_{end})$ checks that all cores required for *cur* are available during τ to τ_{end},
- the available bandwidth at test source v_a is checked to see if: $bw(v_a) > bw(cur) + bw_{alloc}(v_a, \tau, \tau_{end})$ and
- the available memory test source v_a is checked to see if: $mem(v_a)>mem(cur)+mem_{alloc}(v_a, \tau, t_{end})$.

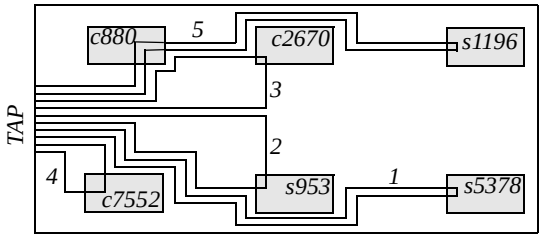

Figure 13. TAM design using our heuristic on System S.

Then the placement of the test resources is checked. If the test resources are on-chip resources and not placed in the system, they are placed at core c_i. If they are floor-planned, a check is performed to determine if they are to be moved.

When the placement of the test resources for the selected test is determined, the test access mechanism is designed and routed. The basic question is if existing wires can be used or new wires must be added. If no routed connection is available connecting all required cores, the distance for adding a completely new connection is re-calculated due to a possible moving of test resources.

Examples of the produced results from the algorithm using System S [5] (Table 1) are the TAM design as in Figure 13 and the test schedule as in Figure 14. The TAM wires 1 to 5 in Figure 13 correspond to the TAM 1 to 5 in Figure 14. For instance, b_5 is the BIST test of core indexed 5 (s5378) and e_5 is the external test of s5378 (note that the BIST tests b_i do not require a TAM but they are placed in Figure 14 to illustrate when they are scheduled).

The computational complexity for the above algorithm, where the test access mechanism design is excluded in order to make it comparable with other approaches, comes mainly from sorting the tests and the two loops. The sorting can be performed using a sorting algorithm at $O(n \times log\ n)$. The worst case occurs when in each loop iteration for the loops only one test is scheduled giving a complexity:

$$\sum_{i=0}^{|P-1|} (P-i) = \frac{n^2}{2} + \frac{n}{2}$$

The total worst case execution time is $n \times log + n^2/2 + n/2$, which is of $O(n^2)$. For instance, the approach by Garg *et al*. [11] and by Chakrabarty [5] both has a worst case complexity of $O(n^3)$.

6. Simulated Annealing

In this section we outline the Simulated Annealing (SA) technique and describe its adoption to be used for scheduling and TAM design. The technique proposed by Kirkpatrick *et al*. [16] uses a hill-climbing mechanism to avoid getting stuck in at local optimum.

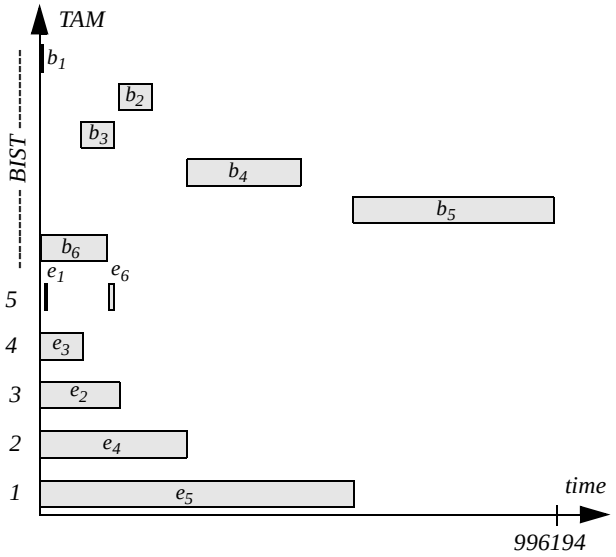

Figure 14. TAM schedule on System S using our heuristic.

```
1:   Construct initial solution, x^now;
2:   Initial Temperature: T=TI;
3:   while stop criteria not met do begin
4:       for i = 1 to TL do begin
5:           Generate randomly a neighboring solution
                 x'∈N(x^now);
6:           Compute change of cost function
                 ΔC=C(x')-C(x^now);
7:           if ΔC≤0 then x^now=x'
8:               else begin
9:                   Generate q = random(0, 1);
10:                  if q<e^(-ΔC/T) then x^now=x'
11:              end;
12:      end;
13:      Set new temperature T=α×T;
14:  end;
15:  Return solution corresponding to the minimum
         cost function;
```

Figure 15. Simulated Annealing algorithm.

6.1 The Simulated Annealing Algorithm

The SA algorithm (Figure 15) starts with an initial solution and a minor modification creates a neighbouring solution. The cost of the new solution is evaluated and if it is better than the previous, the new solution is kept. A worse solution can be accepted at a certain probability, which is controlled by a parameter referred to as temperature.

The temperature is decreased during the optimization process, and the probability of accepting a worse solution decreases with the reduction of the temperature value. The optimization terminates when the temperature value is approximately zero.

6.2 Initial Solution and Parameter Selection

We use our heuristic described in Section 5 with an initial sorting of the tests based on *power* (using *time* and *power×time* results after optimization in the same cost) within the integrated test framework (Section 5) to create the initial solution [17,19]. An example of an initial solution produced for System S is in Figure 13 and Figure 14.

The parameters, initial temperature *TI*, the temperature length *TL* and the temperature reduction factor α ($0<\alpha<1$) are all determined based on experiments.

6.3 Neighbouring Solution in Test Scheduling

In the case when only test scheduling is considered, *i.e.* the TAM is not considered or it is fixed and seen as a resource, we create a neighbouring solution by randomly selecting a test from an existing schedule and schedule it as soon as possible but not at the same place as it was in the original schedule. For instance, creating a neighbouring solution given a test schedule as in Figure 6 with resource graph as in Figure 1 and test compatibility graph as in Figure 2, we randomly select a test, let say t_1. We try to schedule t_1 as soon as possible but not with the same starting time as it had while fulfilling all constraints. Test t_1 was scheduled to start at time 0 and no new starting point exists where constraints are fulfilled until end of t_3 where t_1 now is scheduled. In this case, the test time increases after the modification (getting out of a possible local minimum), however, only temporarily since in the next iteration a test may be scheduled at time 0 (where t_1 used to be).

6.4 Neighbouring Solution in Test Scheduling and TAM Design

When both the test time and the TAM design are to be minimized, a neighbouring solution is created by randomly adding or deleting a wire and then the tests are scheduled on the modified TAM.

If the random choice is to add a wire, a test is randomly selected and a wire is added from its required test source to the core where the test is applied and from the core to the test sink for the test. For instance, if e_3 in System S (Table 1) is selected, a wire is added from the TAP to core c7552 and from core c7552 to the TAP.

If the random choice is to delete a wire, a similar approach is applied. However, a check is performed to make sure that all tests can be applied.

6.5 Cost function

The cost function of a test schedule, *S*, and the TAM, *A*, is:

$$C(S, A) = \beta_1 \times T(S) + \beta_2 \times L(A)$$

where: $T(S)$ is the test application time for a sequence of tests, *S*, $L(A)$ is the total length of the TAM, β_1, β_2 are two designer-specified constants used to determine the importance of the test time and the TAM.

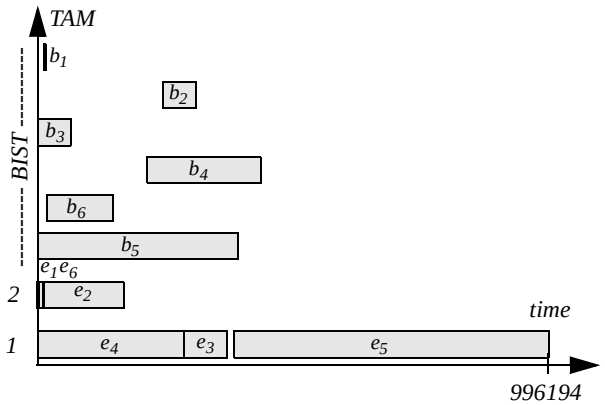

Figure 16. Test schedule on System S using SA.

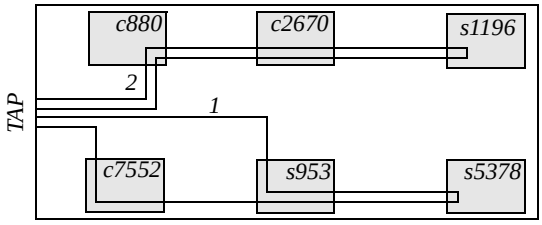

Figure 17. TAM design using SA on System S.

The test application time, $T(S)$, for a schedule, *S*, is:

$$T(S) = \{t_{end}(t_i) \mid \forall t_i(max\{t_{end}(t_i)\}), t_i \in S\}$$

and the length, $L(A)$, of the TAM, *A*, is given by:

$$\sum_{w_j \in A} \sum_{j=0}^{|w_i - 1|} dist(v_j, v_{j+1}), v_j, v_{j+1} \in w_i$$

For the test schedule, S, produced by SA for System S (Figure 16) the test time, T(S) is 996194 (the end time of test e_5) and the length of the TAM (Figure 17) is 160. Comparing this to the results produced by our heuristic [17] shows that test time is the same while the TAM is reduced from 320 (Figure 13) to 160 (Figure 17).

7. Experimental Results

7.1 Benchmarks

We have used the System S [5], which has test conflicts (Table 1) while all other benchmarks and designs have test conflicts and power constraints like the ASIC Z design presented by Zorian [27] with added data made by Chou *et al.* [9] (see the floor-plan in Figure 18) and we added the placement (*x*,*y*) coordinates [17]. For instance, RAM2 is placed at (10,20), which means that the centre of RAM2 has x-coordinate 10 and y-coordinate 20. The design is fully BISTed with a power dissipation limit at 900 mW.

We have also used an extended ASIC Z design where each core is tested by two test sets (one external test and one BIST) and an interconnection test to a neighbouring core [19]; in total 27 tests.

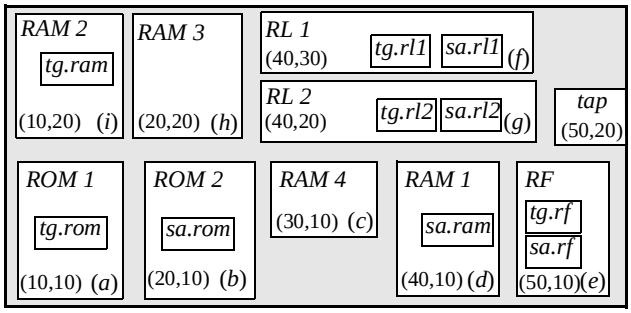

Figure 18. ASIC Z floor-plan.

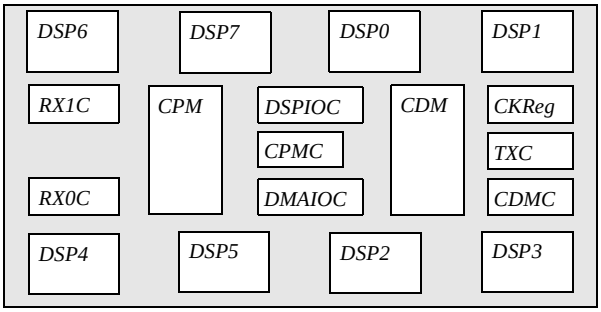

Figure 19. The Ericsson design.

Test	Block	Test	Test time	Test power	Test port
Block-level tests	A	Test A	515	379	scan
	B	Test B	160	205	testbus
	C	Test C	110	23	testbus
	E	Test E	61	57	testbus
	F	Test F	38	27	testbus
	I	Test I	29	120	testbus
	J	Test J	6	13	testbus
	K	Test K	3	9	testbus
	L	Test L	3	9	testbus
	M	Test M	218	5	testbus
Top-level tests	A	Test N	232	379	fp
	N	Test O	41	50	fp
	B	Test P	72	205	fp
	D	Test Q	104	39	fp

Table 2. Characteristics for the industrial design.

We have also used a design with 10 tests presented by Muresan *et al*. [23] and an industrial design with characteristics given in Table 2. The power limitation for the industrial design example is 1200 mW and only one test may use the test bus or the functional pins (fp) at a time. Furthermore block-level tests may not be scheduled concurrently with top-level tests.

The largest design is the Ericsson design [18,19] consisting of 8 DSP cores plus additional logic cores and memory banks, see Figure 19. Design characteristics are in Table 3 where the following notations are used:

- n: DSP core ($0 \le n \le 7$),
- i: common program memory (CPM) bank ($0 \le i \le 7$),
- j: common data memory (CDM) bank ($0 \le j \le 9$),
- l: local data memory (LDM) at a DSP core ($0 \le l \le 3$),
- m: local memory (LZM) bank at a DSP core ($0 \le m \le 1$).

All logic blocks in the Ericsson design are tested by two test sets, one using external tester and one using on-chip resources, while memories are tested with one test set; in total 170 tests.The test access port may be used by more than one test concurrently. However, the other test resources cannot be used concurrently. Furthermore, only one test set may be applied concurrently to each block and maximal power dissipation is 5125 mW. For the implementation, we have simplified our system model (Section 3) regarding the TAM wire design and only allowing a single wire per test.

Block	Test number	Test time	Test Power	Test source	Test sink
RX0C	1	970	375	TAP	TAP
	2	970	375	TG_0	TRA_0
RX1C	3	970	375	TAP	TAP
	4	970	375	TG_0	TRA_0
DSPIOC	5	1592	710	TAP	TAP
	6	1592	710	TG_0	TRA_0
CPMC	7	480	172	TAP	TAP
	8	480	172	TG_0	TRA_0
DMAIOC	9	3325	207	TAP	TAP
	10	3325	207	TG_0	TRA_0
CKReg	11	505	118	TAP	TAP
	12	505	118	TG_0	TRA_0
CDMC	13	224	86	TAP	TAP
	14	224	86	TG_0	TRA_0
TXC	15	364	140	TAP	TAP
	16	364	140	TG_0	TRA_0
CPM_i	17+i	239	80	TG_1	TRA_1
CDM_j	25+j	369	64	TG_1	TRA_1
DSP_n LPM	35+17×n	46	16	$TG_{n,0}$	$TRA_{n,0}$
LDM_l	36+17×n+l	92	8	$TG_{n,0}$	$TRA_{n,0}$
LZM_m	40+17×n+m	23	2	$TG_{n,0}$	$TRA_{n,0}$
$Logic_0$	17×n+42	4435	152	TAP	TAP
	17×n+43	4435	152	$TG_{n,1}$	$TRA_{n,1}$
$Logic_1$	17×n+44	4435	152	TAP	TAP
	17×n+45	4435	152	$TG_{n,1}$	$TRA_{n,1}$
$Logic_2$	17×n+46	7009	230	TAP	TAP
	17×n+47	7009	230	$TG_{n,1}$	$TRA_{n,1}$
$Logic_3$	17×n+48	7224	250	TAP	TAP
	17×n+49	7224	250	$TG_{n,1}$	$TRA_{n,1}$
$Logic_4$	17×n+50	7796	270	TAP	TAP
	17×n+51	7796	270	$TG_{n,1}$	$TRA_{n,1}$

Table 3. The Ericsson design characteristics.

When discussing about our algorithm we use *our*1, *our*2 and *our*3, which corresponds to the initial sorting based on *test power*(*p*), *test time*(*t*) and *test power*×*test time*(*p*×*t*).

Unless stated, we use partitioned testing with run to completion, for the cost function (Section 6.5) $\beta_1=\beta_2=1$ and we have used a Sun Ultra10, 450 MHz CPU, 256 MB RAM.

7.2 Test Scheduling

We have compared our algorithm (nonpartitioned testing) with the nonpartitioned testing approaches proposed by Zorian [27] and Chou *et al.* [9]. We have used the same assumptions as Chou *et al*. and the results are in Table 4. Our approaches (our1, our2, and our3) results, in all cases, in a test schedule with three test sessions (*ts*) at a test time of 300 time units, which is 23% better than Zorian's approach and 9% better than the approach by Chou *et al.*

ts	Zorian		Chou *et al.*		Our1, Our2, Our3	
	Cores	Time	Cores	Time	Cores	Time
1	RAM1, RAM4, RF	69	RAM1,RAM3, RAM4, RF	69	RL2, RL1, RAM2	160
2	RL1, RL2	160	RL1, RL2	160	RAM1,ROM1, ROM2	102
3	RAM2,RAM3	61	ROM1, ROM2, RAM2	102	RAM3, RAM4, RF	38
4	ROM1, ROM2	102				
Test time:		392		331		300

Table 4. ASIC Z test scheduling.

In System S, no power constraints are given and therefore only our2 can be used. Our approach finds using partitioned testing with run to completion after 1 second the optimal solution; see Table 5 (first group).

The results on the industrial design are in Table 5 (second group) where the industrial designer's solution is 1592 time units while our test scheduling achieve a test time of 1077 time units, the optimum, in all cases (our1, our2, and our3), which is 32.3% better than the designer's solution.

The results on design Muresan are in the third group of Table 5. The test time using the approach by Muresan *et al.* is 29 time units and the results using our approaches our1, our2, and our3 are 28, 28 and 26, respectively, all produced within 1 sec. Our SA (TI=400, TL=400, α=0.97) improves to 25 time units using 90 seconds.

When not considering idle power on ASIC Z, the test schedules using our1, our2, and our3 (fourth group in Table 5) all result in a test time of 262. The SA (TI=400, TL=400 and α=0.97) did not find a better solution.

In the experiments considering idle power (fifth group of Table 5), our heuristic approaches our1, our2, and our3 resulted in a solution of 300, 290 and 290, respectively,

Design	Approach	Test time	Diff. to SA/optimum	CPU
System S [5]	Optimal solution	1152180	-	-
	Chakrabarty	1204630	4.5%	-
	Our2 (t)	1152180	0%	1 sec.
Industrial design	Optimal solution	1077	-	-
	Designer	1592	47.8%	-
	Our1 (p)	1077	0%	1 sec.
	Our2 (t)	1077	0%	1 sec.
	Our3 (p×t)	1077	0%	1 sec.
Muresan [23]	SA	25	-	90 sec.
	Muresan [23]	29	16%	-
	Our1 (p)[17]	28	12%	1 sec.
	Our2 (t)[17]	28	12%	1 sec.
	Our3 (p×t)[17]	26	4%	1 sec.
ASIC Z (1)	SA	262	-	74 sec.
	Our1 (p)	262	0%	1 sec.
	Our2 (t)	262	0%	1 sec.
	Our3 (p×t)	262	0%	1 sec.
ASIC Z (2)	SA	274	-	223 sec.
	Our1 (p)[17]	300	10%	1 sec.
	Our2 (t)[17]	290	6%	1 sec.
	Our3 (p×t)[17]	290	6%	1 sec.
Extended ASIC Z (3)	SA	264	-	132 sec.
	Our1 (p)	313	18%	1 sec.
	Our2 (t)	287	9%	1 sec.
	Our3 (p×t)	287	9%	1 sec.
Ericsson	SA	30899	-	3260 sec.
	Our1 (p)	37336	20%	3 sec.
	Our2 (t)	34762	12%	3 sec.
	Our3 (p×t)	34762	12%	3 sec.

Table 5. Test scheduling results.

each produced within 1 second. The SA (TI=400, TL=400 and α=0.99) produced a solution of 274 requiring 223 sec., *i.e.* a cost improvement in the range of 6% to 10%.

The results on Extended ASIC Z when not considering idle power are 313 (our1), 287 (our2) and 287 (our3) (sixth group in Table 5), which were produced by our heuristic after 1 second. The SA optimization (TI=TL=400, α=0.97) produced a solution at a cost of 264 running for 132 seconds, *i.e.* a cost improvement in the range of 9% to 18%.

The results on the Ericsson design (seventh group of Table 5) are 37226, 34762, 34762 produced by our heuristic our1, our2, and our3 (within 3 sec.). The SA algorithm (TI=200, TL=200, α=0.95) produced a solution at 30899 after 3260 seconds.

7.3 Test Resource Placement

In the ASIC Z design all cores have their own dedicated BIST structure. Let us assume that all ROM cores share one

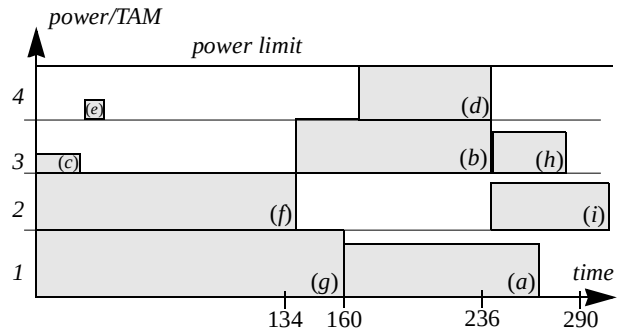

Figure 20. Test schedule for ASIC Z.

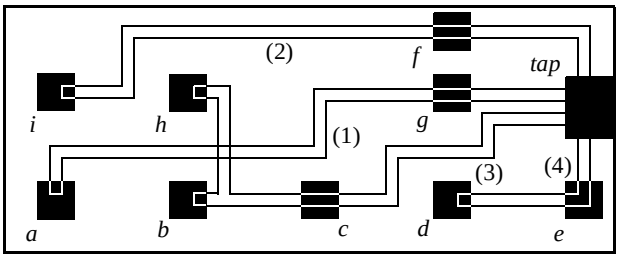

Figure 21. ASIC Z with test data access mechanism.

BIST structure and all RAM memories share another BIST structure; the rest of the cores have their own dedicated BIST structure. Using our placement strategy the test resources in ASIC Z will be placed as in Figure 18.

7.4 Test Access Mechanism Design

Assume for ASIC Z (Figure 18) that all tests are scan-based (1 scan-chain per core) applied with an external tester allowing a maximum of 8 scan chains to operate concurrently. The results using our1, our2, and our3 considering idle power are collected in Table 6. The test schedule and the TAM schedule achieved with heuristic our3 are in Figure 20. The total length of the test access mechanism is 360 units and it is routed as in Figure 21. All solutions were produced within 1 second.

7.5 Test Scheduling and TAM Design

We have made experiments using System S and ASIC Z to demonstrate the importance of integrating test scheduling and TAM design. The ASIC Z is fully BISTed; however, in this experiment we assume all tests are applied using an external tester able of supporting several tests concurrently. We have used two naive approaches, Naive1 and Naive2. Naive1 uses a minimal TAM design connecting all cores in a ring and Naive 2 uses an extensive TAM where each core gets its own dedicated TAM.

For the cost function, we have for ASIC Z used $\beta_1=\beta_2=1$ and for System S $\beta_1=1/1000$ and $\beta_2=1$ and the results are collected in Table 7. Naive 1 produces a low cost TAM design but the test time is long leading to high total cost due

Approach	Test time	TAM cost
Our1 (p)	300	360
Our2 (t)	290	360
Our3 (p×t)	290	360

Table 6. Results on ASIC Z.

Approach	System S			ASIC Z		
	Time	TAM	Total	Time	TAM	Total
Naive 1	1541394	100	1641	699	110	809
Naive 2	996194	360	1356	300	500	800
Our 1 (p)	-	-	-	300	360	660
Our 2 (t)	996194	320	1316	290	360	650
Our 3 (p×t)	-	-	-	290	360	650
SA	996194	160	1156	334	180	514

Table 7. TAM design and test scheduling.

to that all tests have to be scheduled in a sequence. The total cost of using the approach Naive 2 is also high due to the extensive TAM design. Such extensive TAM gives more flexibility, however, other constraints in the design limits its use. Our approaches (our1, our2, and our3) produces better results indicating the importance of considering TAM design and test scheduling in an integrated manner. The TAM design (Figure 13) and the test schedule (Figure 14) achieved using our heuristic for System S (Table 1) have a test time of 996194 and a TAM length of 320 computed within 1 second. The SA (TI=TL=100, α=0.99) was running for 1004 seconds producing a test schedule (Figure 16) and a TAM design (Figure 17) with a test time of 996194 and a TAM length of 160, a TAM improvement of 50%. For ASIC Z, the SA (TI=TL=300, α=0.97) produced after 855 seconds a solution with a cost of 514 (334 for test time and 180 for TAM).

The results comparing our heuristic with SA are collected in Table 8 where, for instance, our heuristic our1 produced a solution for ASIC Z with a total cost of 660 (a test time of 290 and a TAM cost of 360) after 1 second. The test time results are in the range of 10% to 13% better using our fast heuristic (in all cases) compared to the SA optimization. However, the TAM results are much worse using our heuristics and the total cost improvements by the SA are in the range from 21% to 28%.

The results from experiments on Extended ASIC Z are in Table 8 (second group). Our heuristic our1 produced a solution after 1 second with a test time of 313 and a TAM cost of 720, resulting in a total cost of 1033. The solution produced after 4549 seconds by our SA (TI=TL=200, α=0.97) optimization has a test time of 270 and a TAM cost of 560. In this experiment, SA produced a better total cost (range 14% to 24%) as well as better cost regarding test time (range 6% to 16%) and TAM cost (range 18% to 29%). The results on the Ericsson design are collected in Table 8

	Approach	SA	Our1	Our2	Our 3
ASIC Z	Test time	334	300	290	290
	Diff to SA	-	-10%	-13%	-13%
	TAM cost	180	360	360	360
	Diff to SA	-	100%	100%	100%
	Total Cost	514	660	650	650
	Diff to SA	-	28%	21%	21%
	Comp. cost	855 sec.	1 sec.	1 sec.	1 sec.
	Diff to SA	-	-85400	-85400%	-85400%
Extended ASIC Z	**Approach**	**SA**	**Our 1**	**Our 2**	**Our 3**
	Test time	270	313	287	287
	Diff to SA	-	16%	6%	6%
	TAM cost	560	720	660	660
	Diff to SA	-	29%	18%	18%
	Total Cost	830	1033	947	947
	Diff to SA	-	24%	14%	14%
	Comp. cost	4549 sec.	1 sec.	1 sec.	1 sec.
	Diff to SA		-454800%	-454800%	-454800%
Ericsson	**Approach**	**SA**	**Our1**	**Our2**	**Our3**
	Test time	33082	37336	34762	34762
	Diff to SA	-	11%	5%	5%
	TAM	6910	8245	9350	8520
	Diff to SA	-	19%	35%	23%
	Total Cost	46902	53826	53462	51802
	Diff to SA	-	15%	14%	10%
	Comp. cost	15h	81 sec.	79 sec.	62 sec.
	Diff to SA		-66567%	-68254%	-86996%

Table 8. TAM and scheduling results.

(third group). For instance, our heuristic our1 results in a solution with a test time of 37336 and a TAM cost of 8245, which took 81 seconds to produce. The total cost is 53826 when using β_1=1 and β_2=2. The SA (TI=TL=200, α=0.95) optimization produced a solution with a test time of 33082 and a TAM cost of 6910 after 15 hours. In all cases, the SA produces better results. Regarding test time the SA improvement is in the range from 5% to 11%, for the TAM cost in the range from 19% to 35% and the total cost in the range from 10% to 15%.

For all experiments with the SA, the computational cost is extremely higher compared to our heuristics. A finer tuning of the SA parameters could reduce it, however, the extensive optimization is only used for the final design and therefore a high computational cost can be accepted.

8. Conclusions

For complex systems such as SOCs, it is a difficult problem for the test designer to develop an efficient test solution due to the large number of factors involved. The work-flow consists of two consecutive parts: an early design space exploration and an extensive optimization for the final solution. In this paper we have proposed a framework suitable for these two parts where test scheduling, test access mechanism design, test set selection and test resource placement are considered in an integrated manner minimizing the test time and the size of the test access mechanism while satisfying test conflicts and test power constraint. For the early design space exploration, we have implemented an algorithm running at a low computational cost, which is suitable to be used iteratively many times. For the final solution, a more extensive optimization can be justified and we have proposed and implemented a technique based on Simulated Annealing for test scheduling and test access mechanism design.

We have performed several experiments on benchmarks and industrial designs to show the efficiency and usefulness of our approach.

References

[1] J. Aerts and E. J. Marinissen, “Scan Chain Design for Test Time Reduction in Core-Based ICs”, *Proceedings of IEEE International Test Conference (ITC)*, pp. 448-457, Washington, DC, October 1998.

[2] A. Benso, S. Cataldo, S. Chiusano, P. Prinetto, and Y. Zorian, “A High-Level EDA Environment for the Automatic Insertion of HD-BIST Structures”, *Journal of Electronic Testing; Theory and Applications (JETTA)*,Vol.16.3, pp 179-184, June 2000.

[3] H. Bleeker, P. Van Den Eijnden, and F. De Jong, “Boundary-Scan Test: A Practical Approach”, *Kluwer Academic Publishers*, ISBN 0-7923-9296-5, 1993.

[4] K. Chakrabarty, “Test Scheduling for Core-Based Systems Using Mixed-Integer Linear Programming”, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 19, No. 10, pp. 1163-1174, October 2000.

[5] K. Chakrabarty, “Test Scheduling for Core-Based Systems”, *Proceedings of IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, page 391-394, San Jose, CA, November 1999.

[6] K. Chakrabarty, “Design of System-on-a-Chip Test Access Architecture under Place-and-Route and Power Constraints”, *Proceedings of ACM/IEEE Design Automation Conference. (DAC)*, pp. 432-437, Los Angeles, CA, June 2000.

[7] K. Chakrabarty, “Optimal Test Access Architectures for System-On-A-Chip”, *ACM Trans. on Design Automation of Electronic Systems*, vol. 6, pp. 26-49, January 2001.

[8] K. Chakrabarty, “Design of System-on-a-Chip Test Access Architectures Using Integer Linear Programming”, *Proceedings of IEEE VLSI Test Symposium (VTS)*, pp. 127-134, Montreal, Canada, April 2000.

[9] R. Chou, K. Saluja, and V. Agrawal, “Scheduling Tests for VLSI Systems Under Power Constraints”, *IEEE Transactions on VLSI Systems*, Vol. 5, No. 2, pp. 175-185, June 1997.

[10] G. L. Craig, C. R. Kime, and K. K. Saluja, “Test Scheduling and Control for VLSI built-in-self-test”, *IEEE Transactions on Computers*, Vol. 37, No. 9, pp. 1099-1109, September 1988.

[11] M. Garg, A. Basu, T. C. Wilson, D. K. Banerji, and J. C. Majithia, “A New Test Scheduling Algorithm for VLSI Systems”, *Proceedings of the CSI/IEEE Symposium on VLSI Design*, pp. 148-153, New Delhi, India, January 1991.

[12] S. Gerstendörfer and H.-J. Wunderlich, “Minimized Power Consumption for Scan-Based BIST”, *Proceedings of IEEE International Test Conference (ITC)*, pp 77-84, Atlantic City, NJ, September 1999.

[13] G. Hetherington, T. Fryars, N. Tamarapalli, M. Kassab, A. Hassan, and J. Rajski, “Logic BIST for Large Industrial Designs: Real Issues and Case Studies”, *Proceedings of IEEE International Test Conference (ITC)*, pp. 358-367, Atlantic City, NJ, September 1999.

[14] V. Iyengar and K. Chakrabarty, “Precedence-Based, Preemptive, and Power-Constrained Test Scheduling for System-on-a-Chip”, *Proceedings of IEEE VLSI Test Symposium (VTS)*, pp. 368-374, Marina Del Rey, CA, April 2001.

[15] IEEE P1500 Web site. http://grouper.ieee.org/groups/1500.

[16] S. Kirkpatrick, C. Gelatt, M. Vecchi, “Optimisation by Simulated Annealing”, *Science*, Vol. 220, No. 4598, pp. 671-680, 1983.

[17] E. Larsson and Z. Peng, “An Integrated System-On-Chip Test Framework”, *Proceedings of the Design, Automation and Test in Europe Conference (DATE)*, Munchen, Germany, pp 138-144, March 2001.

[18] E. Larsson, Z. Peng and G. Carlsson, “The Design and Optimization of SOC Test Solutions”, *Proceedings of IEEE/ ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 523-530, San Jose, CA, November 2001.

[19] E. Larsson, “An Integrated System-Level Design for Testability Methodology”, *Ph. D. thesis no. 660*, Linköpings universitet, Sweden 2000.

[20] E. J. Marinissen, R. Arendsen, G. Bos, H. Dingemanse, M. Lousberg, and C. Wouters, “A Structured and Scalable Mechanism for Test Access to Embedded Reusable Cores”, *Proceedings of IEEE International Test Conference (ITC)*, pp 284-293, Washington, DC, October 1998.

[21] E. J. Marinissen, Y. Zorian, R. Kapur, T. Taylor, and L. Whetsel, “Towards a Standard for Embedded Core Test: An Example”, *Proceedings of IEEE International Test Conference (ITC)*, pp. 616-627, Atlantic City, NJ, September 1999.

[22] E. J. Marinissen, S. K. Goel, and M. Lousberg, “Wrapper Design for Embedded Core Test”, *Proceedings of IEEE International Test Conference (ITC)*, pp. 911-920, Atlantic City, NJ, October 2000.

[23] V. Muresan, X. Wang, V. Muresan, and M. Vladutiu, “A Comparison of Classical Scheduling Approaches in Power-Constrained Block-Test Scheduling”, *Proceedings of IEEE International Test Conference (ITC)*, pp. 882-891, Atlantic City, NJ, October 2000.

[24] M. Nourani and C. Papachristou, “An ILP Formulation to Optimize Test Access Mechanisms in System-On-A-Chip Testing”, *Proceedings of IEEE International Test Conference (ITC)*, pp 902-910, Atlantic City, NJ, October 2000.

[25] M. Sugihara, H. Date, and H. Yasuura, “A Test Methodology for Core-Based System LSIs”, *IEICE Transactions on Fundamentals*, Vol. E81-A, No. 12, pp. 2640-2645, December 1998.

[26] N. Weste and K. Eshraghian, “Principles of CMOS VLSI Design”, *Addison-Wesley*, ISBN 0-201-53376-6, 1993.

[27] Y. Zorian, “A distributed BIST control scheme for complex VLSI devices”, *Proceedings of the IEEE VLSI Test Symposium (VTS)*, pp. 4-9, Atlantic City, NJ, April 1993.

Erik Larsson received his M.S. and Ph.D. degrees in Computer Systems from Linköping University in 1994 and 2000, respectively. Since October 2001, he is on leave from his Assistant Professor position at the Computer Science department at Linköping University for a Research Fellow position at Fujiwara Laboratory, Nara Institute of Science and Technology, Japan, funded by JSPS (Japan Society for the Promotion of Science). His main research interests are system-level design for testability and design of SOC test solutions.

Zebo Peng is Professor of the chair in Computer Systems, Director of the Embedded Systems Laboratory, and Chairman of the Division for Software and Systems at Linköping University. He received his Ph.D. degree in Computer Science from Linköping University in 1987.

Prof. Peng's current research interests include design and test of embedded systems, electronic design automation, design for testability, hardware/software co-design, and real-time systems. He has published over 120 technical papers in these areas and coauthored the book "System Synthesis with VHDL" (Kluwer Academic, 1997). He was corecipient of two best paper awards at the European Design Automation Conferences (EURO-DAC) in 1992 and 1994. He has served on the program committee of several IEEE/ACM conferences and workshops and was the General Chair of the 6th IEEE European Test Workshop.