An Integrated System-On-Chip Test Framework

Erik Larsson and Zebo Peng

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

Abstract[1]

In this paper we propose a framework for the testing of system-on-chip (SOC), which includes a set of design algorithms to deal with test scheduling, test access mechanism design, test sets selection, test parallelization, and test resource placement. The approach minimizes the test application time and the cost of the test access mechanism while considering constraints on tests, power consumption and test resources. 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. Experimental results shows the efficiency and the usefulness of the proposed technique.

1. Introduction

The increasing complexity of digital systems has led to the need of extensive testing and long test application times. It is therefore important to schedule the tests as concurrently as possible and to design an access mechanism for efficient transportation of test data in the system under test.

When developing the test schedule, conflicts and limitations must be carefully considered. For instance, the 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 on test scheduling. For the test designer, it is also important to get an early impression on the systems overall test characteristics in order to develop an efficient test solution.

Research has been going on in developing techniques for test scheduling, test access mechanism design and testability analysis. For example, a technique to help the designer determine the test schedule for SOC with Built-In Self-Test (BIST) is proposed by Benso *et al.* [1]. In this paper, we combine and generalize several approaches in order to create a framework for SOC testing where:

- tests are scheduled to minimize the test time,
- a test access mechanism is designed and minimized,
- test sets for each block with test resource are selected,
- test resources are floor-planned, and
- tests are parallelized (*i.e.* long scan-chains are divided into several scan-chains of shorter length).

Furthermore, the above tasks are performed under test, power consumption and test resource constraints.

The rest of the paper is organised as follows. After an overview of related work in Section 2, a system modelling technique is introduced in Section 3. Factors affecting the test scheduling and an algorithm which takes them into account in test scheduling and test access mechanism design are then presented in Section 4 and 5, respectively. The paper is concluded with experimental results and conclusions in Section 6 and 7.

2. Related Work

Zorian proposes a test scheduling technique for fully BISTed systems where test time is minimized while power constraints are considered [2]. In order to reduce the complexity of the test controller, tests are scheduled in sessions where no new tests are allowed to start until all tests in a session are completed. Furthermore, tests at blocks placed physically close to each other are grouped in the same test session in such a way that the same control line can be used for all tests in a group. The advantage is that the routing of control lines is minimized.

In a fully BISTed system, each block has its own dedicated test generator (test source) and its own test response evaluator (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.* propose a test scheduling technique where test time is minimized for systems with test conflicts [3] and for core-based systems a test scheduling technique is proposed by Chakrabarty [4]. Chou *et al.* propose an analytic test scheduling technique where test conflicts and power constraints are considered [5]. Another test scheduling approach is proposed by Muresan *et al.* where constraints among tests and power consumption are also considered [6]. In the latter approach, favour is given to reduce the test time by allowing new tests to start

1. This work has partially been supported by the Swedish National Board for Industrial and Technical Development (NUTEK).

even if all tests in a session are not completed. The draw-back is the increasing complexity of the test controller. Note also that in the approaches by Chou *et al.* and by Muresan *et al.*, the systems to be tested are not restricted to fully BISTed systems.

The conflicts among tests can be reduced by using a wrapper such as Boundary scan [7], TestShell [8] or P1500 [9]. These techniques are all developed to increase test isolation and to improve test data transportation.

Usually, several test sets can be used to test a block in the system under test. Sugihara *et al*. propose a technique for selecting test sets where each block may be tested by one test set from an external tester and one test set from a dedicated test generator for the block [10].

The effect on test application time for systems tested by one test set per core using various design styles for test access with the TestShell wrapper is analysed by Aertes *et al.* [11]. Furthermore, the impact on test time using scan-chain parallelization is also analyzed by Aertes *et al.* [11].

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 [12]. External testers can usually only support a maximum of 8 scan chains [12], resulting in long test application time for large designs. Furthermore an external tester's memory is limited by its size.

3. System Modelling

An example of a system under test is given in Figure 1 where each core is placed in a wrapper in order to achieve efficient test isolation and to ease test access. Each core consists of at least one block with added DFT technique and in this example all blocks are tested using the scan technique. The test access port (*tap*) is the connection to an external tester and the test resources, *test generator* 1, *test generator* 2, *test response evaluator* 1 and *test response evaluator* 2, are implemented on the chip.

The system is tested by applying several set of tests where each test set is created at some test generator (source) and the test response is analysed at some test response evaluator (sink).

The system in Figure 1 can be modelled as a *design with test*, $DT = (C, R_{source}, R_{sink}, p_{max}, T, source, sink, core, block, constraint, memory, bandwidth)$, where:

$C = \{c_1, c_2,..., c_n\}$ is a finite set of cores; each core consists of a finite set of blocks, $c_i = \{b_{i1}, b_{i2},..., b_{nm}\}$. Each core consists of at least one block and each block $b_{ij} \in B$ is characterized by:

$p_{idle}(b_{ij})$: idle power,
$par_{min}(b_{ij})$: minimal parallelization degree, and
$par_{max}(b_{ij})$: maximal parallelization degree;

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

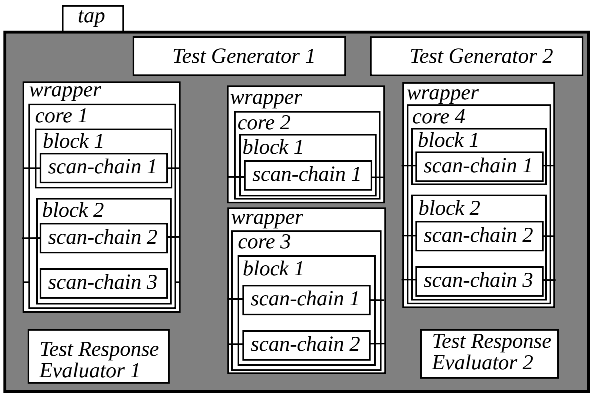

Figure 1. An illustrative example.

$R_{sink} = \{r_1, r_2,..., r_q\}$ 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 *block tests* (*BT*). And each block, b_{ij}, is associated with several block tests, BT_{ijk} (k=1,2,...,l). Each test t_i is characterized by:

$t_{test}(t_i)$: test time at parallelization degree 1, $par(t_i)$=1,
$p_{test}(t_i)$: test power at parallelization degree 1, $par(t_i)$=1,
$t_{memory}(t_i)$: 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: $B \rightarrow C$ gives the core where a block is placed;
block: $T \rightarrow$B gives the block where a test is applied;
constraint: $T \rightarrow 2^B$ gives the set of blocks required for a test;
memory(r_i): memory available at test source $r_i \in R_{source}$;
bandwidth(r_i): bandwidth at test source $r_i \in R_{source}$.

In the above definitions, test time, t_{test}, test power consumption, p_{test}, idle power, p_{idle}, and memory requirement, t_{memory}, are given for each of the tests. The maximal and minimal degree of parallelization for a test is given by par_{max} and par_{min} which determine how much a scan-chain may be divided. For instance, if $par_{max}(b_{31})$=2 and $par_{min}(b_{31})$=1 for block 1 at core 3 in Figure 1, then the scan flip-flops are connected into a single scan-chain ($par(b_{31})$=1) or two scan-chains ($par(b_{31})$=2).

4. The SOC Test Issues

In this section the different issues considered by our SOC test framework are discussed.

4.1 Test Scheduling

Scheduling the tests means that the start time and end time for each test is determined in order to satisfy all constraints. In our approach, the test bus used to transport the test data is also determined by the scheduling algorithm. The basic difference of our scheduling approach compared to previously proposed approaches is illustrated in Figure 2. In the approaches by Zorian [2] and Chou *et al.* [5] no new tests are allowed to start until all tests in a session are

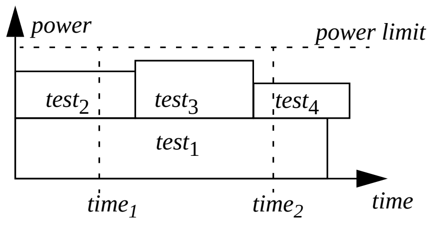

Figure 2. Example of test scheduling.

completed. In their approaches test$_3$ and test$_4$ would not be allowed to be scheduled as in Figure 2. However, in the approach proposed by Muresan *et al*. [6], test$_3$ is allowed to be scheduled as in Figure 2 if it is completed no later than test$_1$. It means that test$_4$ is still not allowed to be started before *test*$_1$ finishes.

In our approach it is optional if tests may start before all tests in a session are completed or not. If it is allowed, test$_3$ and test$_4$ can be scheduled as in Figure 2, which gives more flexibility, but entails usually a more complex test controller.

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

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

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

$$t_{end}(t_i) = t_{start}(t_i) + t_{test}(t_i).$$

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

Let the Boolean function *scheduled*(t_i, *time$_1$*, *time$_2$*) be true if test t_i is scheduled in such a way that the test time overlaps with the time interval [*time$_1$*, *time$_2$*], *i.e.*,

$$\{t_i \in S \wedge \neg(t_{end}(t_i) < time_1 \vee t_{start}(t_i) > time_2)\}.$$

An example to illustrate the function *scheduled* for a set of scheduled tests is shown in Figure 3.

The Boolean function *scheduled*(r_i, *time$_1$*, *time$_2$*) is true if a source r_i is used by a test t_j between *time*$_1$ and *time*$_2$, *i.e.*:

$$\{\exists t_j \in S | r_i = source(t_j) \wedge scheduled(t_j, time_1, time_2)\}.$$

A similar definition is used if a sink r_i is scheduled (used by any test) between *time$_1$* and *time$_2$*.

The Boolean function *scheduled*(*constraint*(t_i), *time$_1$*, *time$_2$*) is true if:

$$\{\exists t_j \in S | block(t_j) \in constraint(t_i) \wedge$$
$$scheduled(t_j, time_1, time_2)\}.$$

The Boolean function *scheduled*(w_i, *time$_1$*, *time$_2$*) is true when a wire w_i is used between *time$_1$* to *time$_2$*:

$$\{\exists t_j \in S | w_i \in bus(t_j) \wedge scheduled(t_j, time_1, time_2)\},$$

where *bus*(t_j) is the set of wires allocated for test t_j.

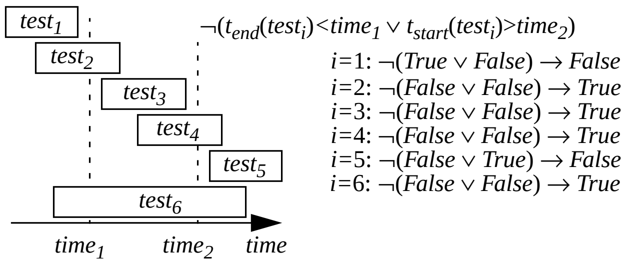
test1
test2
test3
test4
test5
test6
¬(tend(testi)<time1 ∨ tstart(testi)>time2)
i=1: ¬(True ∨ False) → False
i=2: ¬(False ∨ False) → True
i=3: ¬(False ∨ False) → True
i=4: ¬(False ∨ False) → True
i=5: ¬(False ∨ True) → False
i=6: ¬(False ∨ False) → True
time1
time2
time

Figure 3. The function *scheduled*.

4.2 Power Dissipation

In this paper, an additive model used by Zorian [2], Chou *et al.* [5] and Muresan *et al.* [6] for power consumption is assumed. Let p_{sch}(*time$_1$*, *time$_2$*) denote the peak power between *time$_1$* to *time$_2$*, *i.e.*:

$$max\left\{ \sum_{\forall t_i scheduled(t_i, time)} p_{test}(t_i) - p_{idle}(block(t_i)) + \sum_{\forall b_{i,j} \in B} p_{idle}(b_{ij}), time \in [time_1, time_2] \right\},$$

where *scheduled*(t_i, *time*)=*scheduled*(t_i, *time*, *time*).

As an example, applying the function p_{sch}(*time$_1$*, *time$_2$*) on the schedule for a system with 4 tests as in Figure 2, with *time$_1$* and *time$_2$* as indicated in the figure, returns p_{test}(*test$_1$*) + p_{test}(*test$_3$*) + p_{idle}(*block*(*test$_2$*))+p_{idle}(*block*(*test$_4$*)) since it gives the peak power consumption between *time$_1$* and *time$_2$*.

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

4.3 Test Source Limitations

A test generator 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 [12].

The function *memory$_{alloc}$*(r_i, *time$_1$*, *time$_2$*) gives the peak allocated memory between *time$_1$* and *time$_2$* for a given source r_i, *i.e.*:

$$max\left\{ \sum_{\forall t_j scheduled(t_j, time) \wedge r_i = source(t_j)} t_{memory}(t_j), time \in [time_1, time_2] \right\}.$$

A test resource may have a limited bandwidth. For instance, external tester may only support a limited number of scan chains at a time or there could be a limit in the available pins for test. This information is given in the attribute bandwidth for each test resource.

The function *bandwidth$_{alloc}$*(r_i, *time$_1$*, *time$_2$*) gives the maximal number of buses allocated between *time$_1$* and *time$_2$* for a given source r_i, *i.e.*:

$$max\left\{ \sum_{\forall t_j scheduled(t_j, time) \wedge r_i = source(t_j)} |t_{bus}(t_j)|, time \in [time_1, time_2] \right\}.$$

4.4 Test Floor-planning

In the general case it is not feasible to assume that all cores can be tested with only one BIST structure. A block may be tested by several test sets produced and analyzed at different test resources. Furthermore, test resources may be shared among several blocks at different cores. It is therefore important to consider the routing of the test data access mechanism. And an efficient placement of test resources in the system under test must be created in order to minimize the routing cost associated with the test access mechanism.

4.5 Test Set Selection

Each test set is defined by a test source and a test sink. For a test set, its test power consumption, test memory requirement and test application time are defined as discussed in Section 3. We assume that an arbitrary number of test sets can be used to test a block.

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 and test memory requirement. This information should be taken into account in our algorithm.

4.6 Test Parallelization

The test time for a test may be reduced if it is parallelized. This is because dividing a scan-chain into several scan-chains of shorter length will shorten the test application time. Formulas for calculating the test time for scan-based designs are defined by Aertes *et al.* [11]. Similar to Aertes *et al*. we assume that the scan-chain may be divided into equal portions. To simplify the problem, the degree of *parallelization* is assumed to be linear with respect to test time and test power consumption. The test time $t'_{test}(t_i)$ for a test t_i after parallelization is given by:

$$t'_{test}(t_i) = \left\lceil \frac{t_{test}(t_i)}{par(block(t_i))} \right\rceil,$$

where $t_{test}(t_i)$ is the test time when parallelization=1 and $par(block(t_i))$ is the degree of parallelization for the block where t_i is applied.

Assuming that the product *time*×*power* is constant, we let the test power $p'_{test}(t_i)$ for a test t_i after parallelization be given by:

$$p'_{test}(t_i) = p_{test}(t_i) \times par(block(t_i)),$$

where $p_{test}(t_i)$ is the test power when parallelization=1.

The parallelization at a block can not be different for different test sets; the original scan-chain can not be divided into n chains at one moment and to m chains at another moment where $m \neq n$. The function $par(b_{ij})$ denotes the common parallelization degree at block b_{ij}.

4.7 Test Access Mechanism

A test infrastructure transports, and controls the transportation of, test data in the system under test. It transports test patterns from test sources to the blocks and the test response from the blocks to the test sinks.

The test designer faces mainly two problems, namely:

- designing and routing the test access mechanism and
- scheduling the test data transportation.

The system can be modelled as a directed graph, $G=(V,A)$, where V consists of the set of blocks, B, the set of test sources, R_{source}, and the set of test sinks, R_{sink}, *i.e.* $V=B \cup R_{source} \cup R_{sink}$.

An arc $a_i \in A$ between two vertices v_i and v_j indicates a test access mechanism (a wire) where it is possible to transport test data from v_i to v_j. Initially no test access mechanism exists in the system, *i.e.* $A=\emptyset$. However, if the functional infrastructure may be used, it can be included in A initially.

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

1. through the logic of core c_i using the transparent mode of the core;
2. through an optional bypass structure of core c_i; and
3. around core c_i where the access mechanism is not connected to the core.

The advantage of alternatives 1 and 2 above is that the test access mechanism can be reused. However, a delay may be introduced when the core is in transparent mode or its by-pass structure is used. A test wrapper such as the TestShell has a clocked by-pass structure and the impact on the test time using it is analyzed by Aertes *et al.* [11].

In the following, we assume that by-pass may be solved by a non-delay mechanism or that the delay due to clocked by-pass is negligible.

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.

Initially, the test resources may not be placed. In this case, their placement must be determined by our algorithm described in the next section.

The distance between vertex v_i and vertex v_j is given by:

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

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

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

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

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 following 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, which is in the neighbourhood of v_i and has 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 a wire from v_i to v_k; (3)if $v_k = v_j$, terminate otherwise let v_i=v_k and go to (1).

5. The 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 P be an ordered set with the tests ordered based on the key k. If new tests are allowed to be scheduled even if all tests in a session are not completed the function *nexttime*(t_{old}) gives the next time where it is possible to schedule a test:

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

otherwise function *nexttime*(t_{old}) is defined as:

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

The algorithm is depicted in Figure 4 and it 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 block test (BT) for all blocks where all tests within the BT are scheduled. In each iteration of the loop over the tests in P a test *cur* is checked.

If the *parallelization degree* is fixed for the block, *i.e.* some tests have been scheduled for the block, *par*=*par*(b_{ij}) otherwise it is computed:

$$par = min\{par_{max}(b_{ij}), \lfloor (p_{max} - p_{sch}(time, t_{end})) / p(cur) \rfloor,$$
$$bandwidth(v_a, time, t_{end}) - bandwidth_{alloc}(v_a, time, t_{end}),$$

which is the minimum among the available power and the available bandwidth of the test source.

A check is also made to determine if all constraints are fulfilled, *i.e.* it is possible to schedule test *cur* at *time*:

- $\neg\exists t_f$ ($t_f \in BT_{ijk} \wedge t_f \in S \wedge cur \notin BT_{ijk}$) checks that another block test set for current block is not used,
- $par \geq par_{min}(b_{ij})$ checks that the current parallelization degree is larger than the minimal level,
- $\neg$*scheduled*(v_a, *time*, t_{end}) checks that the test source is not scheduled during *time* to t_{end},
- $\neg$*scheduled*(v_c, *time*, t_{end}) checks that the test sink is not scheduled during *time* to t_{end},
- $\neg$*scheduled*(*constraint*(*cur*), *time*, t_{end}) checks that all blocks required for *cur* are not scheduled during *time* to t_{end}, and
- the available memory test source v_a is checked to see if: *memory*(v_a)>t_{memory}(*cur*)+$memory_{alloc}$(v_a, *time*, t_{end}).

```
Sort T according to the key (p, t or p×t) and store the result in P;
S=∅, time=0;
until $\forall b_{pq} \exists BT_{pqr} \forall t_s \in S$ do
    for all cur in P do
        $b_{ij}$=block(cur); $v_a$=source(cur);
        $v_b$=$c_i$; $v_c$=sink(cur);
        par=determine parallelization degree;
        $t_{end}=time+\lceil t_{test}(cur)/par \rceil$;
        $p_{test}(cur)=p_{test}(cur)\times par$;
        if all constraints are satisfied then
            $\neg$scheduled( $v_a$, 0, $t_{end}$) floor-plan $v_a$ at $v_b$;
            $\neg$scheduled( $v_c$, 0, $t_{end}$) floor-plan $v_c$ at $v_b$;
            for all required test resources
                new=length of a new wire $w_j$;
                u=number of wires connecting $v_a$, $v_b$ and $v_c$, and are not
                    scheduled from time to $t_{end}$;
                v=number of wires connecting $v_a$, $v_b$ and $v_c$;
                for all min(v-u,par) $w_j$
                    extend=extend+length of an available wire($w_j$);
                if (par>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 × par})
                        $v_x$, $v_y$=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 $v_b$;
            set parallelization;
            for r = 1 to par
                if there exists a not scheduled wire during time to $t_{end}$
                    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 P;
    time = nexttime(time).
```

Figure 4. The system test algorithm.

Then the placement of the test resources are checked. If the test resources are placed it is checked if they are to be moved.

When the placement of the test resources for the selected test is determined, the corresponding test access mechanism is designed and routed. The basic question is if some existing wires can be used or new wires must be added.

If no routed connection is available connecting all required blocks, the distance for adding a completely new connection is re-calculated due to a possible moving of test resources.

The *extend wire* step in the algorithm extends needed parts to connect the test resources and block with a given wire.

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 for the loops occurs when only one test is scheduled in each iteration resulting in a complexity given by:

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

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.* [3] and by Chakrabarty [4] both have a worst case complexity of $O(n^3)$.

6. Experimental Results

We have performed experiments to show the efficiency of the proposed algorithm.

6.1 Benchmarks

We have used the System S presented by Chakrabarty [4], and ASIC Z design presented by Zorian [2] with added data made by Chou *et al.* [5] (see the floor-plan in Figure 5). We have also used one design consisting of 10 test presented by Muresan *et al.* [6] and an industrial design with characteristics given in Table 1. 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.

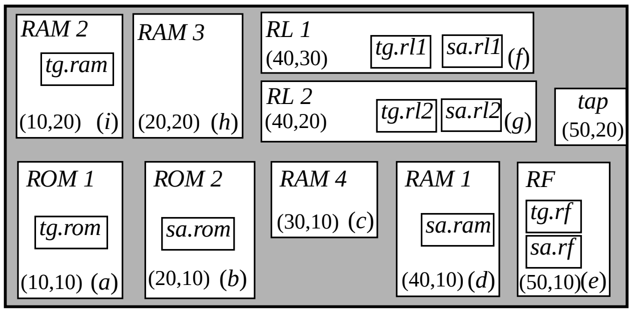

Figure 5. ASIC Z floor-plan.

6.2 Test Scheduling

We have compared our algorithm using initial sorting based on *power*(*p*), *time*(*t*) and *power*×*time*(*p*×*t*) with the approaches proposed by Zorian [2] and Chou *et al.* [5]. We have used the same assumptions as Chou *et al.* and the results are in Table 2. Our approaches 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*.

In System S, no power constraints are given and therefore only test scheduling using initial sorting of tests based on time is performed. Our approach finds the optimal solution, see Table 3.

We have also compared our technique with the technique proposed by Muresan *et al.* [6]. In this case we use the same assumption as Muresan *et al.* which assumes that new tests

	Block	Test	Test time	Idle power	Test power	Test port
Block-level tests	A	Test A	515	1	379	scan
	B	Test B	160	1	205	testbus
	C	Test C	110	1	23	testbus
	E	Test E	61	1	57	testbus
	F	Test F	38	1	27	testbus
	I	Test I	29	1	120	testbus
	J	Test J	6	1	13	testbus
	K	Test K	3	1	9	testbus
	L	Test L	3	1	9	testbus
	M	Test M	218	1	5	testbus
Top-level tests	A	Test N	232	1	379	fp
	N	Test O	41	1	50	fp
	B	Test P	72	1	205	fp
	D	Test Q	104	1	39	fp

Table 1. Characteristics of the industrial design.

can start even if all tests are not fully completed in the current test session. In all cases our technique achieve better solutions, see Table 3.

Finally, the results on an industrial design are in Table 3 where the industrial designer's solution is 1592 time units while our test scheduling achieve a test time of 1077 time units in all sorting variations which is 32.3% better

All solutions using our technique were produced within a second on a Sun Ultra Sparc 10 with a 450 MHz processor and 256 Mbyte RAM.

ts	Zorian		Chou *et al.*		Our algorithm	
	Blocks	Time	Blocks	Time	Blocks	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 2. ASIC Z test scheduling.

6.3 Test Resource Placement

In the ASIC Z design all blocks have their own dedicated BIST structure. Let us assume that all ROM blocks share one BIST structure and all RAM memories share another BIST structure; the rest of the blocks have their own dedicated BIST structure. Using our placement strategy the test resources in ASIC Z will be placed as in Figure 5.

Design	Approach	Test time	Improvement
Chakrabarty's design case [4]	Chakrabarty	1204630	-
	ours(t)	1152810	4.3%
Muresan's design case [6]	Muresan	29	-
	ours(p)	28	3.4%
	ours(t)	28	3.4%
	ours(p×t)	26	10.3%
Industrial design	designer	1592	-
	ours(p)	1077	32.3%
	ours(t)	1077	32.3%
	ours(p×t)	1077	32.3%

Table 3. Results on the designs by Chakrabarty and Muresan as well as the industrial design.

6.4 Test Access Mechanism Design

Assume the floor-planning of ASIC Z as in Figure 5 where each block is placed according to its (x, y) coordinates. For instance, RAM2 is placed at (10,20), which means that the center of RAM2 has x-coordinate 10, and y-coordinate 20. Assume that all tests are scan-based tests applied with an external tester allowing a maximum of 8 scan chains to operate concurrently.

In this experiment we allow a new test to start even if all tests are not completed, see results in Table 4.

The test schedule and the test bus schedule achieved with initial sorting of tests according to power×time and considering idle power is in Figure 7. The total test access mechanism length is 360 units and it is routed as in Figure 6. All solutions were produced within a second on a Sun Ultra Sparc 10 with a 450 MHz processor and 256 Mbyte RAM.

Initial sorting	Test time	Test access mechanism
power	300	360
time	290	360
power×time	290	360

Table 4. Results on ASIC Z.

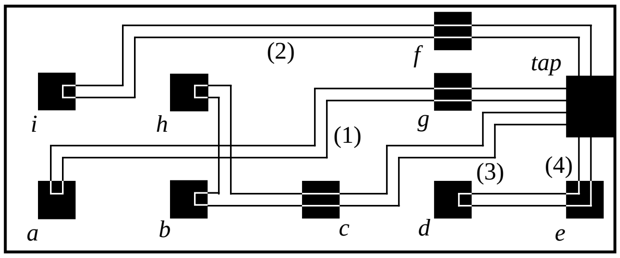

Figure 6. ASIC Z with test data access mechanism.

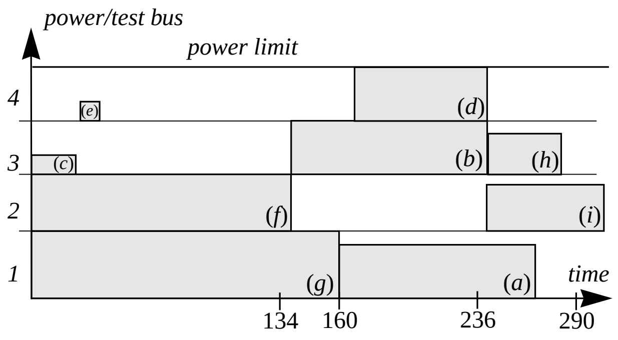

Figure 7. Test schedule for ASIC Z.

7. 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. In this paper we propose a framework where several test-related factors are considered in an integrated manner in order to support the test designer to develop an efficient test solution for a complex system. An algorithm has been defined and implemented, and experiments have been performed to show its efficiency.

References

[1] A. Benso, S. Cataldo, S. Chiusano, P. Prinetto, Y. Zorian, A High-Level EDA Environment for the Automatic Insertion of HD-BIST Structures, *JETTA*,Vol.16.3,pp179-184,June 2000.

[2] Y. Zorian, A distributed BIST control scheme for complex VLSI devices, *Proc. of VLSI Test Symp.*, pp. 4-9, April 1993.

[3] M. Garg, A. Basu, T.C. Wilson, D.K. Banerji, J.C. Majithia, A New Test Scheduling Algorithm for VLSI Systems, *Proc. of the Symp. on VLSI Design*, pp. 148-153, November 1999.

[4] K. Chakrabarty, Test Scheduling for Core-Based Systems, *Proc. of Int. Conf on CAD*, pp. 391-394, January 1991.

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

[6] V. Muresan *et al.*, A Comparison of Classical Scheduling Approaches in Power-Constrained Block-Test Scheduling, *Proc. of Int. Test Conf.*, pp. 882-891, 3-5 October 2000.

[7] H. Bleeker *et al.*, Boundary-Scan Test:A Practical Approach, *Kluwer Academic Publishers*, ISBN 0-7923-9296-5, 1993.

[8] E. J. Marinissen *et al.*, A Structured and Scalable Mechanism for Test Access to Embedded Reusable Cores, *Proc. of International Test Conf.*, pp 284-293, October 18-23, 1998.

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

[10] M. Sugihara, H. Date, H. Yasuura, A Test Methodology for Core-Based System LSIs, *IEICE Trans. on Fund.* vol. E81-A, No. 12, pp. 2640-2645, December 1998.

[11] J. Aerts, E. J. Marinissen, Scan Chain Design for Test Time Reduction in Core-Based ICs, *Proceedings of the International Test Conference*, pp 448-457, 1998.

[12] G. Hetherington *et al.*, Logic BIST for Large Industrial Designs: Real Issues and Case Studies, *Proceedings of the International Test Conference*, pp.358-367, 1999.