A TECHNIQUE FOR TEST INFRASTRUCTURE DESIGN AND TEST SCHEDULING

Erik Larsson and Zebo Peng

Department of Computer and Information Science, Linköping University, Sweden

e-mail: {erila, zpe}@ida.liu.se

Abstract[1]. *We propose a technique for test scheduling and design of test bus infrastructure where test application time and test bus length and width are minimized while constraints on power consumption and test resources are considered. Our approach is suitable for repeated use in the design space exploration process due to its low computational cost. For the final design, we use simulated annealing to optimize the solution. Our technique has been implemented and experimental results show the efficiency of our approach.*

1. Introduction

Test bus infrastructure design and test scheduling are becoming more crucial due to the increasing complexity of digital designs. The system-on-chip (SOC) technique allows a designer to embed different types of cores on a single chip. Each core in the system and the interconnections are usually made testable by the use of appropriate design-for-test (DFT) techniques. Some parts of the system may be tested using an external tester while other parts use Built-In Self-Test (BIST).

When BIST is used, a test bus infrastructure is usually added to the system in order to transport test data from test generators to cores and test response from cores to signature analyzers. However, such an infrastructure gives an increased over-head. Therefore, it must be carefully designed. Furthermore, as the tests for the cores can be scheduled concurrently it may lead to a high power consumption which may damage the system; a good test scheduling method is needed.

Zorian proposes a dedicated test controller for a BIST system [2]. A more general test scheduling algorithm is proposed by Chou *et al.* where test application time is minimized while power consumption is kept under control[1]. A framework for test scheduling for BIST systems is proposed by Benso *et al.* [8] and Chakrabarty proposes a mixed-integer linear programming model for test scheduling [9].

Wrappers such as the Boundary scan technique [5], the P1500 [6] and the TestShell [7] can be placed around the cores to ease test isolation and test access. The wrapper can be controlled so that test data is addressed to the block or by-passed. This allows several blocks to be connected to the same test bus as in TestShell [7].

We propose a technique to combine test scheduling and test bus infrastructure design. Our technique minimizes test application time and test bus length and width while considering constraints on power consumption, BIST resources and cores. Our approach considers both internal tests of the cores as well as interconnection tests between the cores. Furthermore, each core may be tested by several tests using different test facilities where the BIST resources may be shared among several tests at different cores. The low computational cost of our technique allows it to be used repetitively in the design space exploration. For the final design, we use simulated annealing, a more extensive optimization technique.

The rest of the paper is organized as follows. Preliminaries are given in Section 2 and the test scheduling and test bus infrastructure algorithm is in Section 3. In Section 4 the simulated annealing algorithm is presented and we describe our implementation in Section 5. Finally, the paper is concluded with experimental results in Section 6 and conclusions in Section 7.

2. Preliminaries

We assume a DFT flow where an architecture is created from a system-level specification. It is made testable by introducing DFT techniques. Then the tests are scheduled and a test bus infrastructure is added.

If the design, at any step in the flow, shows unaccepted results, some design steps have to be repeated. The design iterations could be repeated many times and it is therefore important that the algorithms used in the design space exploration process has a low computational cost. On the other hand, before the final design is generated, a more extensive optimization can be allowed to take longer time, *i.e* have a higher computational cost.

The system is assumed to use a test infrastructure strategy as proposed by Marinissen *et al.* [7]. It consists of TestShell, the wrapper, TestRail, the test transport mechanism and, Test Control Mechanism, the use to control the test infrastructure. A system can be tested by external testers or with BIST techniques or a combination of both. For instance if we use an external tester for a scan-based design we have to transport test vectors from a test access port to the scan chains and the test response has to be transported from the scan-chains to a test access port. Several tests can be scheduled concurrently if the width of the test bus allow it.

On the other hand if a test at a core uses BIST, the test vectors have to be transported from its test generator to the core and from the core to its signature analyzer. The test generator and signature analyzer can be placed any-

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

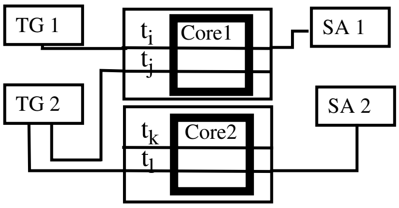

Figure 1: A small illustrative example. For each test a test bus, a test generator (TG) and a signature analyzer (SA) is needed.

```
1: constraints(test,time,schedule)
2: return_value = false;
3:  if (avail_power+testpower<Max power)
4:   if (Signature_Analyzer(test)=free)
5:      if (Test_Generator(test)=free)
6:         if (Core_constraint(test)=free )
7:            return_value = true;
8: return return_value;
```

Figure 3: Constraint checking.

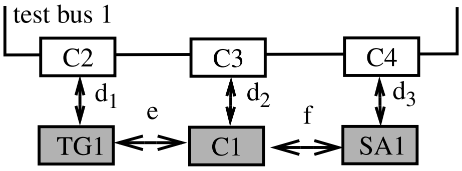

Figure 5: Extending a test bus.

```
1: Sort the tests according to
   power consumption;
2: Place the tests in a list with the
   test consuming most power first;
3: while not empty list
4:    cur = first in list;
5:    while not at end of list do begin
6:       pick cur from list;
7:       if constraints = true then
8:          bus=test_bus(cur,time,schedule)
9:          schedule(cur, time, bus)
10:         remove_from_list(cur);
11:      end;
12:      cur = next_unit;
13:   end;
14:   time=time+min(scheduled units);
15: end;
```

Figure 2: Test scheduling algorithm.

```
1: test_bus( test, time, schedule){
2: if exists_free_bus( time )
3:    if exists_free_used_bus( time )
4:       select the shortest;
5:    else
6:       select the shortest;
7: else
8:    select bus requiring
      shortest extension
9:    until test connected
10:      extend shortest part;
11:      update shortest bus part;
```

Figure 4: Algorithm for Test bus design.

where in the system which is desirable if several cores share them. In Figure 1, an example is shown to illustrate the test bus structure and the sharing of test buses and BIST resources. For instance, *core1* has two tests, t_i and t_j. Test t_i needs test vectors from test generator *TG1* and the test response is evaluated in signature analyzer *SA1*. In the example, test t_i and t_j can not be scheduled concurrently due to the sharing of *SA1*.

If several tests are designed for a certain core there could be interference between them if they are scheduled concurrently. If such interference exists at a core, we say that there exists a core constraint on these tests. This could occur at interconnection test between two cores, if the test is designed so that no other tests can be performed concurrently at these cores. If we assume that the width of all test buses in Figure 1 are equal to 1, test t_k and t_l can not be scheduled concurrently since they share the bus to *core2*.

3. Test Scheduling and Bus Design Algorithms

Our algorithm for combined test scheduling and test bus design is based on the extension of our previously proposed test scheduling algorithm [4] and is illustrated in Figure 2. In general the algorithm works as follows. All tests in the system are sorted and placed in a list according to power consumption. A loop at step 3 is repeated until all tests are scheduled. In each iteration, the test consuming most power is checked and if it can be scheduled according to the power, BIST and core constraints it is removed from the list. A test bus is then selected or designed and the test is scheduled.

If a test can not be scheduled due to constraints, the algorithm checks the next test in the list. If no test can be scheduled at current time, the time is increased and a new iteration starts. At step 7 in Figure 2 power, BIST and core constraints are checked for a test to check if it can be scheduled, the checking procedure is in Figure 3.

A bus is usually a set of parallel wires connecting a set of cores where the number of wires are at least one. However, here we assume that all buses are of width 1. This does not constraint the test bus infrastructure since several buses can be placed next to each other forming a bus with width larger than 1.

The test bus design is performed at step 8 in Figure 2 and the test bus design algorithm is presented in Figure 4. Among the not scheduled buses at a certain time, where all of the test buses connecting all resources for the selected test, the test bus that has the shortest length and has been used before in the schedule is selected. If none of the test buses has been used before, the shortest is selected.

If no test bus is free for scheduling and connecting all resources in the test, a test bus is extended or a new is added. Among all buses in the system, the one that requires the shortest extension is selected.

The example in Figure 5 where a part of a system is shown is used to illustrate step 8 to step 11 of the test bus design algorithm in Figure 4. In the example *C1,...,C4*, are cores and *TG1* is a test generator and *SA1* is a signature analyzer. Let us assume a test at core *C1* is about to be scheduled which uses the BIST resources *TG1* and *SA1* (no core constraints are specified). A test bus connecting *TG1* with *C1* for transporting test vectors from *TG1* to *C1* and a test bus connecting *C1* and *SA1* for transporting the test response from *C1* to *SA1* is needed.

```
1: Construct initial solution, x^now.
2: Initial Temperature: T := TI;
3: For i = 1 to TL do
4:    Generate randomly a neighboring
         solution x′ ∈ N( x^now )
5:    Compute change of cost function
         ΔC := C( x′ ) – C( x^now)
6:    if ΔC ≤ 0 then x^now = x′ else
7:       Generate q := random( 0, 1);
8:       if q < e^(–ΔC/T) then x^now = x′
9:  Set new temperature T := α * T;
10: If stop criterium is not met goto 3.
11: Return solution that corresponds to
    the minimum cost function.
```

Figure 6: Simulated annealing algorithm.

```
// Max power:
250
//core plac.  test  idle test test TG  SA   core
//name x   y  name  pwr  pwr  time
core1  10 10 test1 10   100  125  TG1 TAP NONE
core1  10 10 test2 20   200   75  TAP TAP core2
core2  10 20 test3 50   150   50  TAP SA1 NONE
core2  10 20 test4 50   100  100  TG1 SA1 NONE
core3  10 30 NONE
TC     20 10 NONE
TG1    20 20 NONE
SA1    20 30 NONE
//
```

Figure 7: A design specification accepted by our design tool.

In this example there is no bus connecting *TG1*, *C1* and *SA1*. Therefore a new test bus or an existing has to be extended to include *TG1, C1* and *SA1*. For all test buses, the one requiring the minimal extension is selected.

We assume that a test bus is a chain of buses. For instance, when test bus 1 is used to test core *C3*, both test patterns and test response are transported on test bus 1 concurrently by using the different sections of bus 1. If test patterns arrive from core *C2*, test response is transported in to core *C4*. This reduces the test application time. However, it makes the bus extension more complicated.

Let us assume that test bus 1 is extended along d_1, e and d_3. Even if *TG1*, *C1* and *SA1* now are connected to test bus 1, tests patterns and test response can not be transported concurrently on test bus 1. There is only a single wire along e. When a test bus is extended, this must be considered. A return bus may have to be added.

In the example test bus 1 is selected since there is no other test bus where $2*d_1+2*d_2+2*d_3$ is less then for test bus 1 and $e+f > 2*d_1+2*d_2+2*d_3$. The shortest extension is for *TG1* to connect to *C2*, for *C1* to connect to *C3* and for *SA1* to connect to *C4*. Let us assume that $d_1<d_2<d_3$, then test bus 1 is extended from *C2* to *TG1* and back to *C2* first since d_1 requires the shortest extension. A check is made to see if this extension has changed anything regarding *d2* and *d3*. If for instance $e < d2$, the bus from *TG1* to *C1* is extended and then back to *TG1*. Finally if $f > d3$ an extension is made along *d3*.

4. Simulated annealing

Simulated annealing randomly selects a neighboring solution which is accepted if it is better then the previous. A worse solution can be accepted with certain probability which depends on a control parameter called temperature [10]. The algorithm is presented in Figure 6.

We assume a system given with a set of floor planned blocks and tests specified for each block. The initial test bus and test scheduling will be the one generated by the algorithms described in Section 3 or can be generated randomly. A neighboring solution is generated by adding or deleting a test bus and creating a schedule randomly.

In the search for the optimal solution we minimize a cost function while keeping control on the constraints. We define a cost function $C(x) = T(x) + \beta \times L(x)$ for a solution x as in, where $T(x)$ is the total test application time, $L(x)$ is the total test bus length and, β is a factor defined by the designer.

We terminate the optimization when the temperature reach 0. Experiments guided our selection of initial temperature (*TI*), temperature length (*TL*) and α (cooling rate).

5. Test Bus Design and Test Scheduling Tool

We have implemented our combined test scheduling and test infrastructure design algorithm and a simulated annealing program for optimizing the test scheduling and the test bus design.

Our tool accepts a design specification as in Figure 7. The maximal allowed power consumption is specified and for each core several tests can be defined. Each test is placed at a core and an idle power consumption, a test power consumption and a test application time is specified. For each test, several constraints can be defined.

For instance, *core1* is placed at (10, 10) and to test the core, two tests, *test1* and *test2* are defined. The idle power consumed when *test1* is not applied is 10 mW and when *test1* is applied it uses 100 mW. When *test1* is applied it needs BIST resources *TG1* and *TAP* (Test Access Port). Each test is allowed to have two BIST resources and two core constraints. *TAP* is the connection to the external tester and several tests can be scheduled using the *TAP* at the same time (if a free test bus exists). *Test1* and *test2* can be scheduled concurrently regarding BIST resources. However, *test1* and *test2* can not be scheduled concurrently due to the core constraint of *test2*.

Test may be designed in such way that they can not be scheduled concurrently. For instance if several test exists at a core these test could interfere with each others if scheduled concurrently. Another occasion is at interconnection test where the test may require that two cores are available. For instance, *test2* is scheduled when both *core1* and *core2* are available and during this time no other tests using these cores can be scheduled.

Our implementation allows cores to have no tests such as *core3* in our example. Finally, the BIST resources like, *TG1* is placed at (20,10) in Figure 7.

Design	Traditional		Combined		Average Cost Improv.	Simulated annealing			Average Cost Improv.
	Average Cost	Average CPU(s)	Average Cost	Average CPU(s)		Average Cost	Average Cpu(s)	Average Iterations	
ASICZ	980	<1	870	<1	11.2%	662	3543	88000	32.4%
d4	260	<1	242	<1	6.9%	211	30633	88000	18.8%
d9	637	<1	607	<1	4.7%	435	968	88000	31.7%
d16	1235	<1	1212	<1	1.9%	760	3822	88000	38.5%
d25	2477	<1	2383	<1	3.8%	1436	16630	88000	42.0%

Table 1: Experimental results on ASIC Z and randomly created designs.

6. Experimental Results

The objective of our experiment is to compare our approach with a traditional technique which performs test scheduling and test bus design in two separate steps. We also wanted to analyze the complexity of our algorithm and to check if simulated annealing finds a better solution.

We consider systems using external testers and BIST where the BIST structures may be shared among cores. The designs may have interconnection tests, core constraints and each core may be tested by several tests.

In our experiments we have used ASIC Z [2] and randomly generated designs. For all experiments, we have for the simulated annealing used *TI*=1000, the *TL*=2000, α=0.90 and β in the cost function is 10. ASIC Z consists of 9 blocks where the estimation of the test length is made by Chou *et al.* [1]. The maximal allowed power dissipation is 900 mW. We modified the ASIC Z in such way that we assume that each core is tested by two tests, one using an external tester and one a BIST. The 5 BIST resources are distributed as follows. The RAM memories share one, the ROM memories share one and RF, RL1 and RL2 have their own dedicated. Each core has one interconnection test with one neighbouring core.

The cost of the solution generated by the traditional approach for ASIC Z is 980 while the combined approach uses 870, an improvement of 11.2%, Table 1. The simulated annealing produces a solution at a cost of 662 which is 32.4% better than the traditional solution. The traditional and combined solutions are found within a second while simulated annealing needs 3543 seconds or 88000 iterations to complete.

We randomly generated designs with 4, 9, 16 and 25 cores. For each size of the designs we randomly generated 5 designs and performed experiments on all of them. The presented results are the average result achieved for each size of design, see Table 1. Design d16, for instance, the average cost is 1235 using the traditional approach and 1212 for the combined approach. The computational cost in both cases is below 1 second. The simulated annealing finds a solution at a cost of 760, an improvement of 38.5% compared to the separate solution. However, it requires 3822 seconds or 88000 iterations to complete. The results in Table 1 shows that the combined algorithm finds a solution at a very low computational cost, less than one second for all designs. Furthermore, the simulated annealing finds a better solution in all cases and improvement is larger as design complexity increases.

7. Conclusions

We have proposed an approach for combined test scheduling and test bus design where the test application time and the test bus infrastructure are minimized while constraints on the power consumption, BIST resources and core constraints are considered. Due to its low computational cost our technique is suitable to be incorporated in the design space exploration process. A more extensive optimization based on simulated annealing is used for the final design. In the experiments we compare our combined approach and the simulated annealing optimization with a traditional approach. The results show that the simulated annealing algorithm finds a better solution for all designs, and both the simulated annealing algorithm and the combined approach produce better results than the traditional approach.

References

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

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

[3] D. Gajski, N. Dutt, A. Wu, S. Lin, High-Level Synthesis, Introduction to Chip and System Design, Kluwer Academic Pub., ISBN 0-7923-9194-2, 1992.

[4] E. Larsson, Z. Peng, An Estimation-based Technique for Test Scheduling, Proc. of Electronic Circuits and Systems, pp 25-28, Bratislava, September, 1999.

[5] H. Bleeker, P. Eijnden, F. Jong, Boundary-scan test, Kluwer Academic Publisher, 1993, ISBN 0-7923-9296-5.

[6] IEEE P1500 General Working Group Website, IEEE P1500 Standards For Embedded Core Test, http://grouper.ieee.org/groups/1500.

[7] E. J. Marinissen, et al, A Structured and Scalable Mechanism for Test Access to Embedded Reusable Cores, Proceedings of ITC October 1998.

[8] A. Benso, S. Cataldo, S. Chiusano, P. Prinetto, HD-BIST: A Hierarchical Framework for BIST Scheduling and Diagnosis in SOCs, Proc. of ITC'99, pp 1038-1044, 1999.

[9] K. Chakrabarty, Test Scheduling for Core-Based Systems, Proc. of ICCAD 1999, pp 391.

[10] S. Kirkpatrick, et al., Optimization by Simulated Annealing, Science 220(4598), pp 671-680, 1983.