Defect Probability-based System-On-Chip Test Scheduling

Erik Larsson, Julien Pouget and Zebo Peng
Embedded Systems Laboratory
Linköpings Universitet, Sweden
{erila, g-julpo, zebpe}@ida.liu.se

Abstract[1]

In this paper we address the test scheduling problem for system-on-chip. Different from previous approaches where it is assumed that all tests will be performed until completion, we consider the cases where the test process will be terminated as soon as a defect is detected. This is common practice in production test of chips. The proposed technique takes into account the probability of defect-detection by a test set in order to schedule the test sets so that the expected total test time will be minimized. It supports different test bus structures, test scheduling strategies (sequential scheduling vs. concurrent scheduling), and test set assumptions (fixed test time vs. flexible test time). Several heuristic algorithms have been developed and experiments performed to demonstrate their efficiency.

1. Introduction

The cost of developing electronic systems is increasing and a significant part of the cost is related to the testing of the systems. One efficient way to reduce the total development cost is therefore to reduce the testing cost. Testing cost reduction can be achieved by minimizing the testing time of the system. An efficient ordering, test scheduling, of the execution of the tests will minimize the total testing time.

The core-based design technique is another approach to reduce the increasing development costs. With such a technique, pre-designed and pre-verified blocks of logic, so called cores, are integrated to a complete system, which can be placed on a single die to form a system-on-chip (SOC). To test a SOC, a test bus is used for the transportation of test data in the system and its organization often has a great impact on the test schedule. SOC test scheduling can be performed assuming: *sequential scheduling*, *i.e.* only one test at a time, or *concurrent scheduling*, with a possibility to execute several tests at the same time. The testing time for the execution of each test set can be *fixed,* or *flexible* where it is possible to adjust it.

In a large volume production test for SOC, an *abort-on-fail* approach is usually used, which means that the test sequence is aborted as soon as a fault is detected. This approach is used to reduce the test application time. With the abort-on-fail assumption, the tests should be ordered in such a way that tests with a high probability to fail are scheduled before tests with a lower probability to fail since this will minimize the average testing time.

In this paper we propose a test scheduling technique based on defect detection probability, can be either collected from the production line or generated based on inductive fault analysis. We have

1. The research is partially supported by the Swedish National Program on Socware.

defined models to compute the expected test time as well as scheduling heuristic taking the defect probabilities into account. We have performed experiments to show the efficiency of the proposed approach.

The rest of the paper is organized as follows. An overview of related work is given in Section 2. Sequential test scheduling is discussed in Section 3 and concurrent test scheduling is described in Section 4. The proposed algorithms are presented in Section 5, and the paper is concluded with experimental results in Section 6 and conclusions in Section 7.

2. Related Work

Test scheduling determines the order of the execution of the test sets for a system. The most common objective is to minimize the test time while considering test conflicts. In SOC systems, where each core is equipped with a wrapper, an interface to the test access mechanism (TAM), test conflict is due to the sharing of the TAM or the test bus. The TAM, used for the transportation of test data, is used to connect the test source, the cores and the test sink. The test source is where the test vectors are generated or stored and the test sink is where the test responses are analyzed or stored. An automatic test equipment (ATE) is an example of a test source and test sink.

A TAM can be organized in different ways, which impacts the test scheduling. An example is the AMBA test bus, which makes use of the existing functional bus, however, the tests have to be scheduled in a sequence [2]. An alternative is the approach proposed by Varma and Bhatia where several test buses are used. The tests on each bus are scheduled in a sequence, however, since several buses are allowed, testing can be performed concurrently [11]. Another approach is the TestRail, which allows a high degree of flexibility [8]. The TestRail approach has recently gained interest and several scheduling techniques for SOCs have been proposed [3,5,7]. The approaches assume that the cores are tested with the scan approach. The objective is to arrange the scan-chains into wrapper chains, which then are connected to TAM wires. Iyengar *et al.* made use of integer-linear programming [5] and Huang *et al.* used a bin-packing algorithm. Both these approaches assume that the tests will always be performed until completion.

Koranne proposed an *abort-on-fail* technique to minimize the average-completion time by scheduling tests with short test time early [7]. For sequential testing, several *abort-on-fail* test scheduling techniques considering the defect probability have been proposed [4,6]. Huss and Gyurcsik made use of a dynamic programming algorithm to order the tests [4]. Milor and Sangiovanni-Vincentelli proposed a technique for the selection and ordering of the test sets [10], which is based on the dependencies between the test sets. For SOC testing with cores in wrappers, however, there is no dependency between the testing of different cores. In the approach proposed by Jiang and Vinnakota the actual fault coverage is extracted from the manufacturing line [6]. The technique minimizes the average completion time by ordering of the tests based on probability of failure.

3. Sequential Test Scheduling

In sequential testing, all tests are scheduled in a sequence, one test at a time. When the *abort-on-fail* approach is assumed, if a defect is detected, the testing process should be terminated at once. In order to account for the case when the test responses are compacted into a single signature after a whole test set is applied, we assume that the test abortion occurs at the end of a test set even if the actual defect is detected in the middle of applying the test set. This assumption is also used in our formula to compute the expected test time. Note, this means that the computational results are pessimistic, or the actual test time will be smaller than the computed one, in the case when the tests are actually aborted as soon as the first defect is detected.

Given a core-based system with n cores, for each core i there is a test set t_i with a test time τ_i and a probability of passing p_i (*i.e.* the probability that test t_i will detect a defect at core i is $1\text{-}p_i$). For a given schedule, the *expected test time* for sequential testing is given by:

$$\sum_{i=1}^{n}\left(\left(\sum_{j=1}^{i}\tau_j\right)\times\left(\prod_{j=1}^{i-1}p_j\right)\times(1-p_i)\right)+\left(\sum_{i=1}^{n}\tau_i\right)\times\prod_{i=1}^{n}p_i \qquad (1)$$

For illustration of the computation of the expected test time, we use an example with four tests (Table 1). The tests are scheduled in a sequence as in Figure 1(a). For test t_1, the expected test time is given by the test time τ_1 and the probability of success p_1, $\tau_1\times p_1=2\times0.7=1.4$. Note if there is only one test in the system, our above formula will give the expected test time to be 2 since we assume that every test set has to be fully executed before we can determine if the test is a successful test or not. The expected test time for the completion of the complete test schedule in Figure 1(a) is:

$\tau_1\times(1-p_1)+(\tau_1+\tau_2)\times p_1\times(1-p_2)+$
$(\tau_1+\tau_2+\tau_3)\times p_1\times p_2\times(1\text{-}p_3)+(\tau_1+\tau_2+\tau_3+\tau_4)\times p_1\times p_2\times p_3\times(1-p_4)+$
$(\tau_1+\tau_2+\tau_3+\tau_4)\times p_1\times p_2\times p_3\times p_4=$
$2\times(1-0.7)+(2+4)\times0.7\times(1\text{-}0.8)+$
$(2+4+3)\times0.7\times0.8\times(1\text{-}0.9)+(2+4+3+6)\times0.7\times 0.8 \times0.9\times(1-0.95)+$
$(2+4+3+6)\times0.7\times0.8\times0.9\times0.95 = 9.5$

As a comparision, for the worst schedule, where the test with highest passing probability is scheduled first, the order will be t_4, t_3, t_2, t_1, and the expected test time is 13.6. In the case of executing all tests until completion, the total test time does not depend on the order, and is $\tau_1+\tau_2+\tau_3+\tau_4=15$.

Core i	Test t_i	Test time τ_i	Probability to pass, p_i
1	t_1	2	0.7
2	t_2	4	0.8
3	t_3	3	0.9
4	t_4	6	0.95

Table 1. Example data.

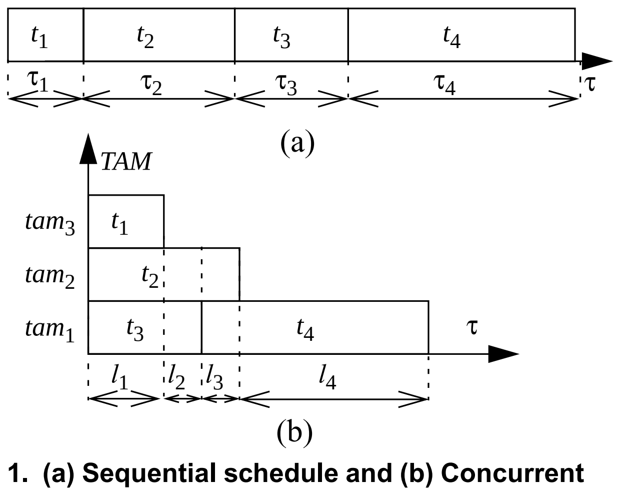

Figure 1. (a) Sequential schedule and (b) Concurrent schedule of the example given in Table 1.

4. Concurrent Test Scheduling

The total test time of a system can be reduced by executing several tests at the same time, *concurrent testing*. Concurrent testing is for instance possible in systems with several test buses. In this section, we analyze concurrent scheduling with fixed test time per test set and flexible test time per test set.

4.1 Test Sets with Fixed Test Times

A concurrent test schedule of the example system used in Section 3 with data as in Table 1 assuming 3 TAMs (test buses) is in Figure 1(b). The test schedule (Figure 1 (b)) consists of a set of sessions,

S_1, S_2, S_3, and S_4. The test session S_1 consists of test t_1, t_2 and t_3; $S_1=\{t_1,t_2,t_3\}$. The length of a session S_k is given by l_k. For instance $l_1=2$. We assume now that the abortion of the test process can occure at any time during the application of the tests. To simplify the computation of expected test time, it is assumed that the test process will terminate at the end of a session (note this is again a pessimistic assumption). The probability to reach the end of a session depends in the concurrent test scheduling approach not only on a single test but on all tests in the session. For instance, the probability to complete session 1 depends on the tests in session 1: t_1, t_2 and t_3. As can be observed in Figure 1(b), only test t_1 is fully completed at the end of session 1. For a test t_i that is not completed at the end of a session, the probability p_{ik} for it to pass all test vectors applied during session k is given by: $p_{ik} = p_i^{l_k/\tau_i}$. It can be seen that for a test set t_i, which is divided into m sessions, the probability that the whole test set is passed is equal to:

$$\prod_{k=1}^{m} p_{ik} = p_i^{\frac{l_1}{\tau_i}} \times p_i^{\frac{l_2}{\tau_i}} \times \ldots \times p_i^{\frac{l_k}{\tau_i}} = p_i^{\sum_{k=1}^{m} \frac{l_k}{\tau_i}} = p_i \quad \text{since} \quad \sum_{k=1}^{m} \frac{l_k}{\tau_i} = \frac{1}{\tau_i} \times \sum_{k=1}^{m} l_k = 1. \qquad (2)$$

For example, the probability for the tests in session S_1 are (Figure 1(b)): $p_{11} = p_1 = 0.7$, $p_{21} = 0.8^{2/4} = 0.89.$, $p_{31} = 0.9^{2/3} = 0.93$. The formula for computing the expected test time for a complete concurrent test schedule is given as:

$$\sum_{i=1}^{n}\left(\left(\sum_{j=1}^{i} l_j\right) \times \left(\prod_{j=1}^{i-1} \prod_{\forall t_k \in S_j} p_{kj}\right) \times \left(1 - \prod_{\forall t_k \in S_i} p_{ki}\right)\right) + \left(\sum_{i=1}^{n} \tau_i\right) \times \prod_{i=1}^{n} p_i \qquad (3)$$

As an example, the computation of the expected test time for the test schedule in Figure 1(b) is given below. First we compute the probability for each test set in each session. The probabilities p_{11}, p_{21}, p_{31} are computed to 0.7, 0.89, and 0.93, respectively (see above). The rest are computed to: $p_{22}=0.8^{1/4}=0.95$, $p_{32}=0.9^{1/3}=0.96$, $p_{23}=0.8^{1/4}=0.95$, $p_{43}=0.95^{1/6}=0.99$, $p_{44}=0.95^{5/6}=0.96$. From the formula we can compute the expected test time for this example as:

$l_1\times(1-p_{11}\times p_{21}\times p_{31})+(l_1+l_2)\times p_{11}\times p_{21}\times p_{31}\times(1-p_{22}\times p_{32})+$
$(l_1+l_2+l_3)\times p_{11}\times p_{21}\times p_{31}\times p_{22}\times p_{32}\times(1-p_{23}\times p_{43})+$
$(l_1+l_2+l_3+l_4)\times p_{11}\times p_{21}\times p_{31}\times p_{22}\times p_{32}\times p_{23}\times p_{43}\times(1-p_{44})+$
$(l_1+l_2+l_3+l_4)\times p_{11}\times p_{21}\times p_{31}\times p_{22}\times p_{32}\times p_{23}\times p_{43}\times p_{44}=$
$2\times(1-0.7\times0.89\times0.93)+(2+1)\times0.7\times0.89\times0.93\times(1-0.95\times0.96)+$
$(2+1+1)\times0.7\times0.89\times0.93\times0.95\times0.96\times(1-0.95\times0.99)+$
$(2+1+1+5)\times0.7\times0.89\times0.93\times0.95\times 0.96\times0.95\times0.99\times(1-0.96)+$
$(2+1+1+5)\times0.7\times0.8\times0.9\times0.95= 5.61$.

As a comparision, if all tests are assumed to be executed until completion, the total test time will be 9.

4.2 Test Sets with Flexible Test Times

A way to further reduce the test application time is to modify, if possible, the test times of the individual test sets. For instance, in scan tested cores the test times at each core can be modified by loading several scan chains in parallel. The scan-chains and the wrapper cells are to form a set of wrapper chains. Each wrapper chain is then connected to a TAM wire. If a high number of wrapper chains are used, their length is shorter and the loading time of a new test vector is reduced. However, the higher number of wrapper chains require more TAM wires.

In Figure 2(a) the TAM bandwidth $|W|$ is 4, there are four wires in $W=\{w_1, w_2, w_3, w_4\}$. The testing of each core is performed by transporting test vectors on the assigned TAM wires to a core and the test response is also transported from the core to the test sink using the TAM. The testing of cores sharing TAM wires cannot be executed concurrently. For instance, the testing of core 1 and core 2 cannot be performed concurrently due to the sharing of TAM wire w_1 (Figure 2(a)). A test schedule for the system is given in Figure 2(b) and the computation of the expected test time can be done using formula 3 in Section 4.

5. Algorithms

The algorithm for test scheduling based on defect probability in the sequential case is straight forward, it sorts the tests in descending order based on $\tau_i \times p_i$ and schedule the tests in this order (Figure 3(a)).

In concurrent scheduling with fixed test times, we sort the tests based on $\tau_i \times p_i$ and select *n* tests for the *n* TAMs based on the sorted list. The selected tests are scheduled and removed from the list. As soon as a test terminates, a new test from the list of unscheduled tests is selected. The process continues until all tests are scheduled (sketch of the algorithm is given in Figure 3(b).)

We have based the algorithm for concurrent test scheduling with flexible test times on the algorithm presented by Flottes *et al.* [1]. The algorithm is outlined in Figure 3(c). We start by computing the test time for all configurations of each core. The Pareto-optimal points are selected and placed descended in a sorted list computed based on the defect level. The algorithm selects a test from the sorted list and tries to maximize the TAM usage of the available TAM wires at each iteration step.

We will use the ITC’02 benchmark Q12710 [9] to illustrate the algorithm. The design consists of 4 scan tested cores. For the computation of test time τ_{ij} for a test t_i at core *i* given *j* TAM wires/ wrapper chains, we have used the wrapper chain algorithm proposed by Flottes *et al.* [1]. The results are in Table 2(a). We have also attached each test with a probability of passing (non failure testing) for each core: core 1 p_1=0.9, core 2 p_2=0.7, core 3, p_3=0.85, core 4 p_4=0.85. For the Pareto-optimal points, we have computed the cost $c_{ij}=\tau_{ij}\times(1-p_i)$ (Table 2(b)).

The test scheduling result using Q12710 at W=12 is presented in Figure 2. The scheduling algorithm starts at τ=0 and selects the test maximizing the TAM usage; test t_2 is selected at configuration *j*=14 (Table 2). At τ=0, all 20 TAM wires are free and t_2 uses 14 (it maximizes the TAM usages). A temporary session length is given by the test time of t_2. We try to find a test that fits the temporary session using no more than 6 TAM wires. From the search in the cost list, we find t_3. We cannot schedule more tests starting at τ=0 and we therefore let $\tau=\tau_{36}$ (the time when test t_3 ends). We then try to schedule a test during the time given by the test time at t_3 minus the test time at t_2. No test is found. We then come to a point where a new session is to be created and we select t_4 with a configuration using 10 wires. We can in the same session schedule the last test; t_1. As soon

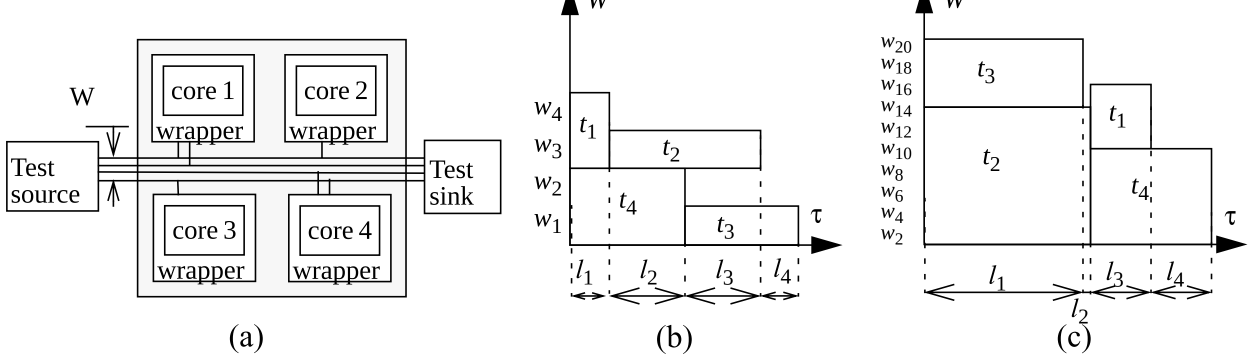

Figure 2. (a) SoC example. (b) SOC schedule of the example (Table 1), (c) schedule of Q12710.

as a test is scheduled, all its configurations are removed from the list *L* (Table 2(b)). When the list is empty, all tests are scheduled and the algorithm terminates.

6. Experimental Results

We have performed experiments using the Q12710 design (presented in Section 5), one of the ITC'02 designs [9]. In the Q12710 design all cores are tested using the scan technique. It means that the test time is not fixed. In the experiments, we have used the following four test scheduling approaches:

1. *Sequential testing with fixed test times* with the test time for each core given by the test time where the product $\tau_i \times j$ (j - number of TAM wires/wrapper chains) is minimal.
2. *Sequential testing with fixed test times* with the fixed test time for each core given by the test time when we assigned the full bandwidth |W| to each core.
3. *Concurrent testing with fixed test times*. The fixed test times in concurrent scheduling of test sets with fixed test times, was computed in the same way as in Sequential scheduling with fixed test times, that is for each core *i* we minimize the $\tau_i \times j$ (j is the number of TAM wires/wrapper chains).
4. *Concurrent testing with flexible test times*. Finally, for the concurrent scheduling of test sets with flexible test times, we computed all test times and selected the Pareto-optimal points (discussed in Section 5).

We have for each of the four approaches assumed a TAM with a bandwidth in the range from 8 to 24 wires in steps of 4 (Table 3). For each approach at each bandwidth we have computed the total

```
1. Compute the cost $c_i = p_i \times \tau_i$ for all tests $t_i$
2. Sort the costs $c_i$ descending in $L$
3. until $L$ is empty (all tests are scheduled)
4. begin
5.    select, schedule and remove
      the first test in $L$
6. end
```

(a)

```
1. Compute the cost $c_i = p_i \times \tau_i$ for all tests $t_i$
2. Sort the costs $c_i$ descending in $L$
3. $f$=number of TAMs
4. $\tau$=0 // current time,
5. until $L$ is empty (all tests are scheduled)
   begin
6.   at time $\tau$ until $f$=0 begin
7.      select tests from L in order and reduce
        $f$ accordingly
8.   end
9.   $\tau$=time when first test terminates.
10. end
```

(b)

```
1. Compute the test time $\tau_{ij}$ for every core $i$ with $j$ in 1 to $|W|$
2. Select the Pareto-optimal points
3. Compute the cost $c_{ij} = \tau_{ij} \times (1-p_i)$ for all Pareto-optimal points.
4. Sort the costs $c_{ij}$ descending in $L1$
5. until all tests are scheduled // L1 is empty begin
6.      for all tests begin
7.         for all created sessions $j$ begin
8.            $\tau_{bk}$ - is the beginning of session $k$.
9.            for all configurations of tests $t_i$ in the list $L1$ begin
10.              select in order the test $t_i$ that maximizes the usage
11.              of available TAM wires
12.           end
13.           if $\tau_{bk} + \tau_i < \tau_{max}$ then begin
14.              schedule $t_i$
15.              remove all configurations of $t_i$ from $L1$
16.           end else store test in $L2$
17.     end
18.     if no test is scheduled then begin
19.        schedule the first test in $L1$
20.     end
21.     move all tests in L2 to L1
22. end
```

(c)

Figure 3. (a) Sequential test scheduling algorithm. (b) Concurrent test scheduling algorithm for tests with fixed test times (c) Concurrent test scheduling algorithm for tests with flexible test times.

test time without considering defect probability and the estimated test time assuming a given defect probability. For instance, in the sequential scheduling (approach 1) at bandwidth 8, the test time is 11566270 and the estimated test time is 9057026, a average reduction of 21.7%. The estimated test time is, as expected, lower than the test time in all cases. It is interesting to observe that in some cases such as in approach 4 at bandwidth 12 and 16, the test time is actually higher at bandwidth 16 compared to bandwidth 12. However, the estimated test time is lower at the bandwidth 16 compared to bandwidth 12.

core i	wire j	test time τ_{ij}	$(\tau_i \times j)$	difference (%)
1	6	951094	5706564	-
1	7	829115	5803805	1.7%
1	2	3000853	6001706	4.2%
1	4	1779357	7117428	24.7%
2	4	6323834	25295336	-
2	7	3680684	25764788	1.9%
2	10	2644464	26444640	4.5%
2	14	2222349	31112886	23.0%
2	2	15743179	31486358	24.5%
3	6	2145671	12874026	-
3	2	6531263	13062526	1.5%
3	4	3733199	14932796	11.1%
3	9	1588751	14298759	16.0%
4	6	2145671	12874026	-
4	2	6531263	13062526	1.5%
4	9	1588751	14298759	11.1%
4	4	3733199	14932796	16.0%

(a)

core i	wire j	test time τ_{ij}	cost c_{ij} $(\tau_{ij} \times (1-p_i))$
2	2	15743179	4722954
2	4	6323834	1897150
2	7	3680684	1104205
3	2	6531263	979689
4	2	6531263	979689
2	10	2644464	793339
2	14	2222349	666705
3	4	3733199	559980
4	4	3733199	559980
3	6	2145671	321851
4	6	2145671	321851
1	2	3000853	300085
3	9	1588751	238313
4	9	1588751	238313
1	4	1779357	177936
1	6	951094	95109
1	7	829115	82912

(b)

Table 2 (a) The Pareto-optimal points and their difference to the optimal for Q12710, (b) The Pareto-optimal points for design Q12710 sorted descending based on the cost $c_{ij}=\tau_\iota \times (1-p_i)$.

7. Conclusions

In this paper we have presented several test scheduling techniques for system-on-chip (SOC) that take into account the defect probability of each test set. The advantage with our approach is that by considering defect probabilities during the test scheduling process, the expected test time can be substantially reduced, which is important in large volume production of SOC where the testing process is terminated as soon as a defect is detected.

We have analyzed several different test bus structures and scheduling approaches, and defined models to compute the expected test times and test scheduling algorithms for several types of test buses. We have also performed experiments to demonstrate the efficiency of our approach.

References

[1] M.L. Flottes, J.Pouget, and B.Rouzeyre, “Sessionless Test Scheme: Power-constrained Test Scheduling for System-on-a-Chip”, *Proceedings of the 11th IFIP on VLSI-SoC*, pp. 105-110, Montpellier, June 2001.

[2] P. Harrod, "Testing reusable IP-a case study", *Proceedings of International Test Conference (ITC),* Atlantic City, NJ, USA, pp. 493-498, 1999.

[3] Y. Huang, W.-T. Cheng, C.-C. Tsai, N. Mukherjee, O. Samman, Y. Zaidan and S. M. Reddy, "Resource Allocation and Test Scheduling for Concurrent Test of Core-based SOC Design", *Proceedings of IEEE Asian Test Symposium (ATS)*, pp 265-270, Kyoto, Japan, November 2001.

[4] S. D. Huss and R. S. Gyurcsik, "Optimal Ordering of Analog Integrated Circuit Tests to Minimize Test Time", *Proceedings of the ACM/IEEE Design Automation Conference (DAC)*, pp. 494-499, 1991.

[5] V. Iyengar, and K. Chakrabarty, and E. J. Marinissen, "Test wrapper and test access mechanism co-optimization for system-on-chip", *Proc. of International Test Conference (ITC)*, Baltimore, MD, USA, pp. 1023-1032, 2001.

[6] W. J. Jiang and B. Vinnakota, "Defect-Oriented Test Scheduling", *IEEE Transactions on Very-Large Scale Integration (VLSI) Systems, Vol. 9, No. 3,* pp. 427-438, June 2001.

[7] S. Koranne, "On Test Scheduling for Core-Based SOCs", *Proceedings of the IEEE International Conference on VLSI Design (VLSID),* pp. 505-510, Bangalore, India, January 2002.

[8] E. J. Marinissen, R. Arendsen, G. Bos, H. Dingemanse, M. Lousberg, C. Wouters, "A structured and scalable mechanism for test access to embedded reusable cores", *Proceedings of International Test Conference (ITC)*, Washington, DC, USA, pp. 284-293, October 1998.

[9] E. J. Marinissen, V. Iyengar, and K. Chakrabarty, "A Set of Benchmarks for Modular Testing of SOCs", *Proceedings of International Test Conference (ITC)*, pp. 519-528, Baltimore, MD, USA, October 2002.

[10] L. Milor and A. L. Sangiovanni-Vincentelli, "Minimizing Production Test Time to Detect Faults in Analog Circuits", *IEEE Trans. on Computer-Aided Design of Integrated Circ. & Sys. ,* Vol. 13, No. 6, pp 796-, June 1994.

[11] P. Varma and S. Bhatia, "A Structured Test Re-Use Methodology for Core-based System Chips", *Proceedings of International Test Conference (ITC),* pp. 294-302, Washington, DC, USA, October 1998.

		Bandwidth					Average
		8	12	16	20	24	
Sequential - fixed test times (minimizing $\tau_i \times j$)	Test time without considering defect probability τ_t:	11566270	11566270	11566270	11566270	11566270	11566270
	Test time considering defect probability τ_e:	9057026	9057026	9057026	9057026	9057026	9057026
	Difference $(\tau_t-\tau_e)/\tau_t \times 100$:	21.7%	21.7%	21.7%	21.7%	21.7%	21.7%
Sequential - fixed test times (τ_i *at* |W| wires)	Test time without considering defect probability τ_t:	11566270	6651081	6228966	6228966	6228966	7380850
	Test time considering defect probability τ_e:	9057026	4818481	4415579	4415579	4415579	5424449
	Difference $(\tau_t-\tau_e)/\tau_t \times 100$:	21.7%	27.6%	29.1%	29.1%	29.1%	26.5%
Concurrent - fixed test times ($\tau_i \times j$)	Test time without considering defect probability τ_t:	11566270	6323834	6323834	6323834	6323834	7372321
	Test time considering defect probability τ_e:	9057026	4653913	4516623	4516623	3667362	5282309
	Difference $(\tau_t-\tau_e)/\tau_t \times 100$:	21.7%	26.4%	28.6%	28.6%	42.0%	28.3%
Concurrent - flexible test times	Test time without considering defect probability τ_t:	11041062	5821966	5955548	3811100	3177502	5961436
	Test time considering defect probability τ_e:	7319653	4073684	4012068	2929196	2508045	4168529
	Difference $(\tau_t-\tau_e)/\tau_t \times 100$:	33.7%	30.0%	32.6%	23.1%	21.0%	30.0%

. Table 3. Experimental results on Q12710 from the ITC'02 benchmarks.