JOURNAL OF ELECTRONIC TESTING: Theory and Applications 21, 651–658, 2005

Abort-on-Fail Based Test Scheduling*

ERIK LARSSON, JULIEN POUGET AND ZEBO PENG
Embedded Systems Laboratory, Department of Computer Science, Linköpings Universitet, Sweden
erila@ida.liu.se

Received March 24, 2004; Revised August 25, 2005

Editor: K.K. Saluja

Abstract. The long and increasing test application time for modular core-based system-on-chips is a major problem, and many approaches have been developed to deal with the problem. Different from previous approaches, where it is assumed that all tests will be performed until completion, we consider the cases where the test process is terminated as soon as a defect is detected. Such abort-on-fail testing is common practice in production test of chips. We define a model to compute the expected test time for a given test schedule in an abort-on-fail environment. We have implemented three scheduling techniques and the experimental results show a significant test time reduction (up to 90%) when making use of an efficient test scheduling technique that takes defect probabilities into account.

Keywords: SoC testing, test scheduling, abort-on-fail, defect probability

1. Introduction

The cost of testing is highly related to the test application time; hence test time minimization is of high importance. In production test of modular core-based system-on-chip (SoC), each core has its dedicated tests, and the tests are ordered by a scheduling algorithm such that the test application time is minimal. In high volume production testing an *abort-on-fail* approach, i.e. the test sequence is aborted as soon as a fault is detected, is common practice as it reduces the test application time. With the abort-on-fail assumption, the tests should, in order to minimize the expected test time, be ordered such that tests with a high probability to fail are scheduled prior to tests with a lower probability to fail.

The ordering of tests, determined by a scheduling algorithm prior to test application, is highly impacted by the test access mechanism (TAM). The TAM, which is responsible for the transportation of test data, can be organized in different ways. The AMBA test bus is a TAM that makes use of the existing functional bus and where the tests are scheduled in a sequence [2]. In TestRail [9] and the approach by Varma and Bhatia [14] several test buses can be used. The test on each test bus is sequential; however, as several buses can be used, concurrent application, several buses being active at the same time, is possible. The TAM design impacts the scheduling alternatives and the following alternatives can be distinguished:

- *sequential scheduling*, i.e. only one test at a time, and
- *concurrent scheduling*, where it is possible to execute several tests concurrently.

Concurrent scheduling can, depending on TAM design, be further divided into the cases with *fixed-width* TAM and those with *flexible-width* TAM [6].

A scheduling algorithm arranges the scanned elements (scan-chain, wrapper inputs and wrapper outputs) at each core into wrapper chains, connects the wrapper chains to TAM wires, and determines a start time and an end time for the testing of each core. For

*The research is partially supported by the Swedish National Program STRINGENT.

test scheduling, Iyengar et al. made use of integer-linear programming [5] and Huang et al. [3] used a bin-packing algorithm. Both approaches minimize the test application time, assuming that all tests will be applied and fully executed. Koranne proposed an *abort-on-fail* technique to minimize the average-completion time by scheduling tests with short test time early [8]. However, defect probability is not taken into account. For non SoCs and sequential testing only, several abort-on-fail test scheduling techniques considering the defect probability have been proposed. 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 tests based on the dependencies between the tests [11]. For modular SoC, where cores are equipped with wrappers, there is usually 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 and the technique minimizes the average completion time by ordering of the tests based on the failure probability assuming sequential testing [7].

In this paper we address SoC test scheduling considering defect detection probability (the defect probability per module can be collected from the production line). We define models to compute the expected test time for a given test schedule and we have performed experiments to show the efficiency of effective test scheduling taking defect probability into account.

The rest of the paper is organized as follows. Expected test time is discussed in Section 2; in Section 2.1 for sequential test scheduling and in Section 2.2 for concurrent test scheduling. The paper is concluded with experimental results in Section 3 and conclusions in Section 4.

2. Expected Test Time Calculation

In this section we discuss expected test time calculation for high volume production test of modular core-based SoCs for the following scheduling approaches; sequential, concurrent with fixed-width TAM, and concurrent with flexible-width TAM.

2.1. Sequential Test Scheduling

In sequential testing, all tests are scheduled in a sequence, one test at a time. The test application time, when all tests are assumed to be executed until completion, is given as the sum of each test's (core's) test time. The test application time for a system as in Fig. 1 with the given sequential test schedule as in Fig. 2 is $\tau_1 + \tau_2 + \tau_3 + \tau_4$. Given the test times for each of the tests in Table 1, the test application time is easily computed to $2 + 4 + 3 + 6 = 15$.

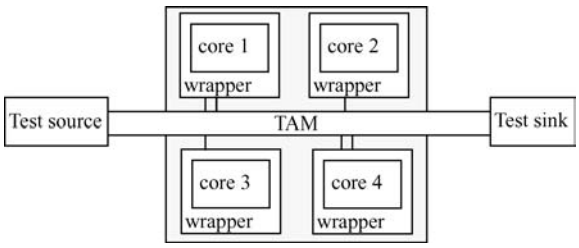

Fig. 1. SoC example.

When all tests have been applied, test evaluation is performed to determine if the chip is faulty or fault-free. Assume the chip testing with a test schedule as in Fig. 2 with a test time of 15, and that a fault is detected by the first test (t_1 with test time (τ_1) equals 2). In such a scenario, 13 time units are spent on testing a faulty chip.

However, when the *abort-on-fail* approach is assumed, the testing is terminated as soon as a defect is detected. If a test schedule is given as in Fig. 2 and the first test detects a fault, the testing is terminated after 2 time units (the testing time of the first test).

In the case when abort-on-fail testing is *not* considered, the ordering in sequential testing is of no importance. The test application time is always the sum of the testing time of each core. However, when

Table 1. Data for an example system.

Core i	Test t_i	Test time τ_i	Pass probability p_i	Cost $\tau_i \times p_i$
1	t_1	2	0.7	1.4
2	t_2	4	0.6	2.4
3	t_3	3	0.9	2.7
4	t_4	6	0.8	4.8

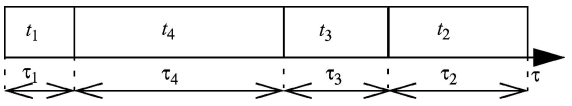

Fig. 2. Sequential schedule of the example (Table 1).

abort-on-fail testing is used, the probability of passing a test impacts the testing times. If a core has a high probability of having a fault, it should be scheduled earlier than the testing of a core that has a low probability of having a fault.

In order to further discuss the expected test time we assume that given is a core-based system with n cores, for each core i there is a test 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 $1 - p_i$). The probabilities are assumed to be given (e.g., collected from the production line).

For a given schedule produced by any test scheduling algorithm, the *expected test time* for sequential testing is given by:

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

To illustrate the expected test time computation, we use an example with four tests (Table 1). The tests are scheduled in a sequence as in Fig. 2. For test t_1, the partial expected test time is given by the test time τ_1 and the probability of success p_1. Note if there is only one test in the system, the formula above will give the expected test time to be 2 since we assume that every test has to be fully executed before we can determine if the test is successful or not.

The expected test time for the completion of the whole test schedule in Fig. 2 is given in Fig. 3.

As a comparison, for the worst schedule, where the test with highest passing probability is scheduled first, the order will be t_2, t_3, t_4, t_1, and the expected test time is 12.1. In the case of executing all tests until completion, the total test time does not depend on the order, and is $\tau_1 + \tau_2 + \tau_3 + \tau_4 = 15$.

2.2. *Concurrent Test Scheduling*

The test time for a system can be reduced by executing several tests at the same time, *concurrent testing*. Concurrent testing can be divided based on TAM architecture into fixed-width TAM and flexible-width TAM. In this section we will discuss expected test time calculation for both fixed-width TAM and flexible-width TAM.

2.2.1. Fixed-width TAM. Fig. 5 shows for the example system (with the test data given in Table 1) a test schedule for the fixed-width TAM architecture in Fig. 4. Note that, in contrast to the sequential case in Section 2.1 where only one test was executed at a time, it is possible to apply multiple tests concurrently. The advantage with concurrent application is that the test application time can be reduced. The total test time for the schedule in Fig. 5 is 9, which is 6 time units less then the test time for the schedule in Fig. 2.

The expected test time calculation for a given test schedule must be extended so that all concurrent tests at any moment are considered. For the calculation, we introduce sessions as illustrated in Fig. 5. The test schedule (Fig. 5) consists of four sessions, S_1, S_2, S_3, and S_4. Test session S_1 consists of test t_1, t_2 and t_3; $S_1 = \{t_1, t_2, t_3\}$, and session S_2 consists of test t_2 and t_3, $S_2 = \{t_2, t_3\}$; hence a test may be included in several sessions. We will assume that a session ends when a test terminates. For example, session S_1 terminates at time point 2 when test t_1 terminates. The length of a session S_k is given by l_k. For instance $l_1 = 2$. Note, that it is possible to set the length of the sessions to a fixed time step as well in order to have a finer granularity.

We assume that the abortion of the test process can occur at any time during the application of the tests. The probability to reach the end of a session depends 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 (S_1): t_1, t_2 and t_3. As can be

$\tau_1\times(1-p_1)+$	2×(1−0.7)+	// test t_1 fails
$(\tau_1+\tau_4)\times p_1\times(1-p_4)+$	(2+6)×0.7×(1-0.6)+	// test t_1 passes but t_4 fails
$(\tau_1+\tau_4+\tau_3)\times p_1\times p_4\times(1-p_3)+$	(2+6+3)×0.7×0.6×(1-0.9)+	// test t_1 and t_4 pass but t_3 fails
$(\tau_1+\tau_4+\tau_3+\tau_2)\times p_1\times p_4\times p_3\times(1-p_2)+$	(2+6+3+2)×0.7×0.6×0.9×(1−0.8)+	// test t_1, t_4 and t_3 pass but t_2 fails
$(\tau_1+\tau_4+\tau_3+\tau_2)\times p_1\times p_4\times p_3\times p_2=$	(2+6+3+2)× 0.7×0.6×0.9×0.8 = 8.2	// all tests pass.

Fig. 3. Expected test time calculation in sequential testing.

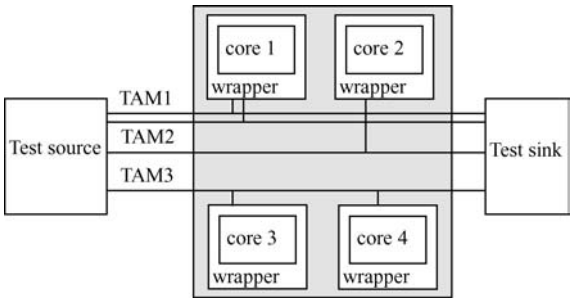

Fig. 4. SoC example.

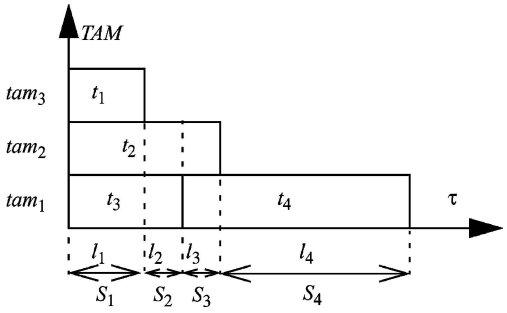

Fig. 5. Concurrent schedule of the example in Table 1.

observed in Fig. 5, only test t_1 is fully completed at the end of session 1. We have in Section 2.1 discussed the expected test time computation when a test is run until completion. Here, we have to define how to handle a test t_i that is not completed at the end of a session. We assume that the probability p_{ik} for a test t_i to pass the test vectors applied during session k (S_k) as:

$$p_{ik} = p_i^{l_k/\tau_i} \tag{2}$$

If a test t_i is divided into m sessions, the probability that the whole test set is passed is equal to:

$$\begin{aligned}\prod_{k=1}^{m} p_{ik} &= p_i^{\frac{l_1}{\tau_i}} \times p_i^{\frac{l_2}{\tau_i}} \times \cdots \times p_i^{\frac{l_k}{\tau_i}} \\ &= p_l^{\sum_{k=1}^{m} \frac{l_k}{\tau_i}} = p_i\end{aligned} \tag{3}$$

since:

$$\sum_{k=1}^{m} \frac{l_k}{\tau_i} = \frac{1}{\tau_i} \times \sum_{k=1}^{m} l_k = 1 \tag{4}$$

The formula for computing the expected test time for a complete concurrent test schedule is given as:

$$\begin{aligned}\sum_{i=1}^{n} \left(\left(\sum_{j=1}^{i} l_j \right) \times \left(\prod_{j=1}^{i-1} \prod_{\forall t_k \in s_j} p_{kj} \right) \right. \\ \left. \times \left(1 - \prod_{\forall t_k \in S_i} p_{ki} \right) \right) + \left(\sum_{i=1}^{n} \tau_i \right) \times \prod_{i=1}^{n} p_i\end{aligned} \tag{5}$$

We make use of the test schedule in Fig. 5 to illustrate how to compute the probabilities in session S_1. The three tests in session S_1 are t_1, t_2, and t_3, and for each test we compute the probability of pass during the length of the session in relation to the test of each test. The session length (l_k) is 2, which for t_1 with test length 2 is: $p_{11} = 0.7^{2/2}$. For test t_1 the probability p_{11} is equal to p_1 as t_1 is run until completion. For test t_2 with a test length of 4 the probability during session 1 is $p_{21} = 0.8^{2/4} = 0.89$, and $p_{31} = 0.9^{2/3} = 0.93$.

The computation of the expected test time for the given test schedule in Fig. 5 is given in Fig. 6. First we compute the probability for each test set in each session, as discussed in the previous paragraph for session S_1. The probabilities p_{11}, p_{21}, p_{31} are computed to 0.7, 0.89, and 0.93, respectively. For the other sessions (S_1, S_2, S_3):

$$\begin{aligned}p_{22} &= 0.7^{1/4} = 0.91. \\ p_{32} &= 0.9^{1/3} = 0.96. \\ p_{23} &= 0.8^{1/4} = 0.95. \\ p_{43} &= 0.95^{1/6} = 0.99. \\ p_{44} &= 0.95^{5/6} = 0.96.\end{aligned}$$

Fig. 6 shows the computation of the expected test time using Eq. (5). The expected time to have a fault at

$l_1 \times (1 - p_{11} \times p_{21} \times p_{31}) +$	$2 \times (1 - 0.7 \times 0.89 \times 0.93) +$	//fault-session 1
$(l_1 + l_2) \times p_{11} \times p_{21} \times p_{31} \times (1 - p_{22} \times p_{32}) +$	$(2+1) \times 0.7 \times 0.89 \times 0.93 \times (1 - 0.91 \times 0.96) +$	//fault-session 2
$(l_1 + l_2 + l_3) \times p_{11} \times p_{21} \times p_{31} \times p_{22} \times p_{32} \times (1 - p_{23} \times p_{43}) +$	$(2+1+1) \times 0.7 \times 0.89 \times 0.93 \times 0.91 \times 0.96 \times (1 - 0.95 \times 0.99) +$	//fault-session 3
$(l_1 + l_2 + l_3 + l_4) \times p_{11} \times p_{21} \times p_{31} \times p_{22} \times p_{32} \times p_{23} \times p_{43} \times (1 - p_{44}) +$	$(2+1+1+5) \times 0.7 \times 0.89 \times 0.93 \times 0.91 \times 0.96 \times 0.95 \times 0.99 \times$	//fault-session 4
$(l_1 + l_2 + l_3 + l_4) \times p_{11} \times p_{21} \times p_{31} \times p_{22} \times p_{32} \times p_{23} \times p_{43} \times p_{44} =$	$(1 - 0.96) + (2+1+1+5) \times 0.7 \times 0.8 \times 0.9 \times 0.95$	//fault-free chip
	$= 5.66.$	//expected test time

Fig. 6. Expected test time calculation in concurrent testing.

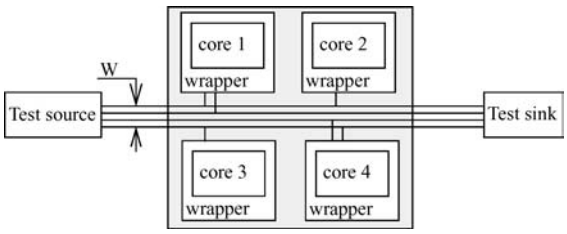

Fig. 7. SoC example.

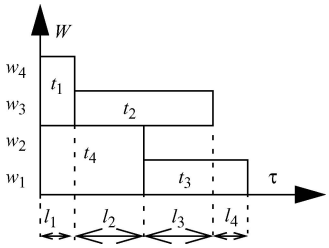

Fig. 8. SoC test schedule for the example Table 1.

the end of each session is computed as well as the expected time that all tests pass. The expected test time for this example is 5.66. As a comparison, if all tests are assumed to be executed until completion, the total test time will be 9.

2.2.2. Flexible-width TAM. A flexible-width TAM architecture with TAM bandwidth |W| 4 for the same example design is shown in Fig. 7. The cores are assigned to one or more of the four TAM wires in W=$\{w_1, w_2, w_3, w_4\}$, A test schedule is given in Fig. 8. Note that the concept of buses is not used here. Instead the TAM width is seen as a set of wires where each wire can be assigned to any core. The constraint is that concurrent testing on a wire is not allowed. For instance, the testing of core 1 and core 2 cannot be performed concurrently due to the sharing of TAM wire w_3.

The test application time when abort-on-fail testing is *not* used is given by the time the TAM wire that is used the most. In the example in Fig. 8 it is wire w_1.

The expected test time can be computed with the same approach as for fixed-width TAM in Section 2.2.1.

3. Experimental Results

The objective with the experiments is to demonstrate that by taking the defect probability into account and allowing abort-on-fail testing, the test time will be reduced, and that the expected test time can be further reduced by taking both defect probability and test scheduling strategy into consideration.

Table 2. Pass probabilities (%) for design d695, p22810 and p93791.

	Design		
Core	d695	p22810	p93791
1	98	98	99
2	99	98	99
3	95	97	97
4	92	93	90
5	99	91	91
6	94	92	92
7	90	99	98
8	92	96	96
9	98	96	91
10	94	95	94
11		93	93
12		91	91
13		92	91
14		93	90
15		99	99
16		99	98
17		99	97
18		95	99
19		96	99
20		97	99
21		93	90
22		99	99
23		96	90
24		98	98
25		99	92
26		92	96
27		91	95
28		91	91
29		93	90
30		94	96

We have compared three scheduling approaches:

- *sequential testing* where the tests are ordered and executed one at a time in a sequence,
- *fixed test time* where the test time for each core is assigned to a fixed test time prior to scheduling (for details see [13]), and

Table 3. Experimental results. Test time ($\tau_{x(n|a)}$)—technique x ($x = 1, 2, 3$), n = *no abort-on-fail*, a = *abort-on-fail*.

		Technique 1—sequential testing			Technique 2—fixed test time			Technique 3—flexible test time			
Design	TAM width	$\tau_{1,n}$	$\tau_{1,a}$	Improvement $(\tau_{1,n} - \tau_{1,a})/\tau_{1,n}$ (%)	$\tau_{2,n}$	$\tau_{2,a}$	Improvement $(\tau_{2,n} - \tau_{2,a})/\tau_{2,n}$ (%)	$\tau_{3,n}$	$\tau_{3,a}$	Improvement $(\tau_{3,n} - \tau_{3,a})/\tau_{3,n}$ (%)	Comparision $(\tau_{1,n} - \tau_{3,a}/\tau_{1,n})$ (%)
d695	128	36158	31113	13.9	13348	10884	18.5	13348	9468	29.1	73.8
	96	36232	31158	14.0	19932	14716	26.2	17257	11712	32.1	67.7
	80	36232	31158	14.0	19932	14881	25.3	18691	14509	22.4	59.9
	64	45798	40586	11.4	32857	25483	22.4	20512	16652	18.8	63.6
	48	45972	40692	11.5	33031	27388	17.1	29106	23983	17.6	47.8
	32	78077	70411	9.8	65136	50998	21.7	41847	33205	20.6	57.5
Average				12.4			21.9			23.4	61.7
p22810	128	503088	423852	15.7	142360	72628	49.0	128332	50484	60.7	89.9
	96	503534	423852	15.8	215339	93921	56.4	159994	59177	63.0	88.2
	80	503635	423993	15.8	223463	122641	45.1	195733	71995	63.2	85.7
	64	531631	443459	16.6	294046	141999	51.7	236186	92218	61.0	82.6
	48	619537	510795	17.6	418226	213995	48.8	352834	121865	65.5	80.3
	32	664665	535586	19.4	574120	355646	38.1	473418	160237	66.1	75.9
Average				16.7			48.2			63.2	83.8
p93791	128	639827	491279	23.1	618150	431628	30.2	457862	124278	72.9	80.6
	96	672119	524537	22.0	650402	488083	25.0	639217	192940	69.8	71.3
	80	1174475	942900	19.7	1155800	852477	24.2	787588	197393	74.9	83.2
	64	1240170	983943	20.7	1221495	922505	24.5	945425	270979	71.3	78.1
	48	1377123	1072900	22.1	1358448	1003672	26.1	1220469	360045	70.5	73.9
	32	2432511	1941982	20.2	2432511	1941892	20.2	1827819	682101	62.7	72.0
Average				21.3			25.0			70.4	76.6

- *flexible test time* where the testing time versus the number of used TAM wires for each core is flexible (for details see [12]).

We have modified the algorithms such that it is possible to set the sorting cost function to consider or not consider defect-probabilities. For each of the three scheduling techniques, we have compared the test time when not taking defect probability into account with taking defect probability into account at various TAM widths. We have used the three ITC'02 designs [10] d695, p22810 and p93791 and for all experiments we have used an AMD 1800 machine (1.53 GHz, 512 MB RAM). The computational cost is usually a few seconds, and it never exceeds 15 seconds. The test-passing probability for each core is collected in Table 2, and the experimental results are collected in Table 3.

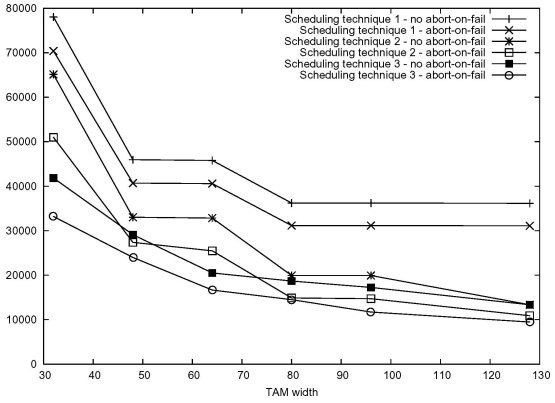

Fig. 9. Results on d695.

The results are also plotted in Fig. 9 (d695), Fig. 10 (p22810) and Fig. 11 (p93971).

The experimental results show that by taking the defect probability into account, the test time can be

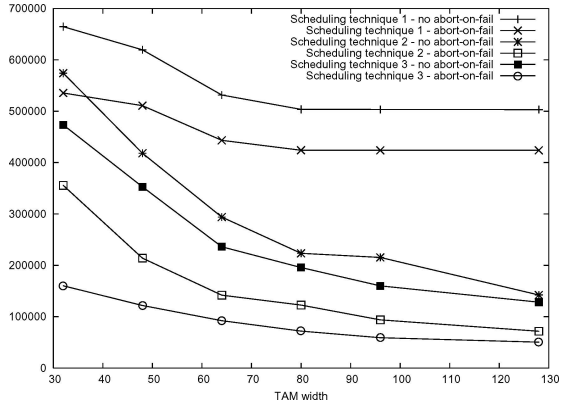

Fig. 10. Results on p22810.

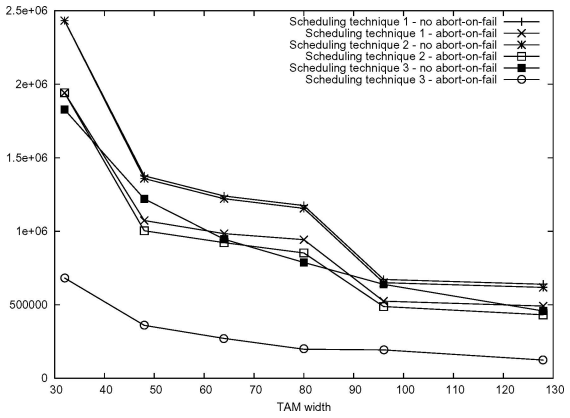

Fig. 11. Results on p93971.

significantly reduced. For instance, the test time is on average reduced for the sequential testing technique, by 12.4% for d695, 16.7% for p22810, and 21.3% for p93971, and, for the flexible testing technique reduced by 23.4% for d695, 63.2% for p22810, and 70.4% for p93971.

The results also show that significant reduction in test time can be achieved by considering both defect probabilities and test scheduling strategy. The test times produced by the sequential testing technique when defect probability is not considered with the results produced by the flexible testing time technique when defect probability is taken into account show that the testing times are reduced on average by 61.7% for d695, 83.8% for p22810, and 76.6% for p93791.

4. Conclusions

A major problem in testing modular core-based SoCs is the long test application times. In this paper we have discussed test scheduling techniques for SoC that take the defect probability of each test into account in the scheduling process, assuming an abort-on-fail test environment. The advantage with our approach is that by considering defect probabilities during the test scheduling process, the expected test time can be minimized, which is important in large volume production of SoC where the testing process is terminated as soon as a defect is detected. We have defined models to compute the expected test time for a varity of TAM architectures and scheduling approaches. We have performed experiments on three scheduling approaches and three benchmarks where we show a significant test time reduction (up to 90%) by taking both defect-probability into account in test scheduling and making use of an efficient test scheduling 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*, Montpellier, 2001, pp. 105–110.
2. P. Harrod, "Testing reusable IP-a case study," in *Proceedings of International Test Conference (ITC)*, Atlantic City, NJ, USA, 1999, pp. 493–498.
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," in *Proceedings of IEEE Asian Test Symposium (ATS)*, Kyoto, Japan, 2001, pp. 265–270.
4. S.D. Huss and R.S. Gyurcsik, "Optimal Ordering of Analog Integrated Circuit Tests to Minimize Test Time," in *Proceedings of the ACM/IEEE Design Automation Conference (DAC)*, 1991, pp. 494–499.
5. V. Iyengar, and K. Chakrabarty, and E.J. Marinissen, "Test Wrapper and Test Access Mechanism Co-Optimization for System-On-Chip," in *Proceedings of International Test Conference (ITC)*, Baltimore, MD, USA, 2001, pp. 1023–1032.
6. V. Iyengar, K. Chakrabarty, and E.J. Marinissen, "Test Access Mechanism Optimization, Test Scheduling, and Tester Data Volume Reduction for System-on-Chip," *Transactions on Computers*, vol. 52, no. 12, pp. 1619–1632, 2003.
7. 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, 2001.
8. S. Koranne, "On Test Scheduling for Core-Based SOCs," in *Proceedings of the IEEE International Conference on VLSI Design (VLSID)*, Bangalore, India, January 2002, pp. 505–510.
9. 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," in *Proceedings of International Test Conference (ITC)*, Washington, DC, USA, 1998, pp. 284–293.
10. E.J. Marinissen, V. Iyengar, and K. Chakrabarty, "A Set of Benchmarks for Modular Testing of SOCs," in *Proceedings*

of International Test Conference (ITC), Baltimore, MD, USA, 2002, pp. 519–528.
11. L. Milor and A.L. Sangiovanni-Vincentelli, "Minimizing Production Test Time to Detect Faults in Analog Circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 13, no. 6, pp. 796– , 1994.
12. J. Pouget, E. Larsson, and Z. Peng, "SOC Test Time Minimization Under Multiple Constraints," in *Proceedings of Asian Test Symposium (ATS)*, Xian, China, 2003.
13. J. Pouget, E. Larsson, Z. Peng, M.L. Flottes, and B. Rouzeyre, "An Efficient Approach to SoC Wrapper Design, TAM Configuration and Test Scheduling," *Formal Proceedings of European Test Workshop 2003 (ETW* '03), Maastricht, The Netherlands, 2003, pp. 51–56.
14. P. Varma and S. Bhatia, "A Structured Test Re-Use Methodology for Core-based System Chips," in *Proceedings of International Test Conference (ITC)*, Washington, DC, USA, 1998, pp. 294–302.