Hybrid BIST Test Scheduling Based on Defect Probabilities

Zhiyuan He, Gert Jervan, Zebo Peng, and Petru Eles
Embedded Systems Laboratory (ESLAB)
Linköping University, Sweden
{g-zhihe, gerje, zebpe, petel}@ida.liu.se

Abstract[1]

This paper describes a heuristic for system-on-chip test scheduling in an abort-on-fail context, where the test is terminated as soon as a defect is detected. We consider an hybrid BIST architecture, where a test set is assembled from pseudorandom and deterministic test patterns. We take into account defect probabilities of individual cores in order to schedule the tests so that the expected total test time in the abort-on fail environment is minimized. Different from previous approaches, our hybrid BIST based approach enables us not only to schedule the tests but also to modify the internal test composition, the order and ratio of pseudorandom and deterministic test patterns, in order to reduce the expected total test time. Experimental results have shown the efficiency of the proposed heuristic to find good quality solutions with low computational overhead.

1. Introduction

The rapid advances of the microelectronics technology in recent years have enabled design and manufacturing of complex electronic systems, where multiple pre-designed and pre-verified blocks, usually referred as cores, are integrated into one single die. Such complex system-on-chip (SoC) designs pose great challenges to the test engineers since inefficient test methodologies may significantly increase the testing time of the system [1]. This consequently increases the total cost of such systems and leads to increased time to market.

To test the individual cores in a SoC a set of test resources, such as test pattern source and sink together with an appropriate test access mechanism (TAM), have to be available [2]. As one of the increasing challenges in testing SoCs is the volume of the test data and its transportation, the organization and amount of these test resources has high importance. Depending on the test architecture, the tests can be scheduled in parallel, i.e. several tests can be executed at the same time, or sequentially, i.e. only one test at a time. In this context scheduling of the test has significant impact on test time.

1. This work has been partially supported by the Swedish Foundation for Strategic Research (SSF) under the Strategic Integrated Electronic Systems Research (STRINGENT) program.

In a production test environment, where a large number of dies have to be tested, an abort-on-fail approach is usually employed. It means that the test process is stopped as soon as a fault is detected. This approach leads to reduced test times and consequently also to reduced production costs, as faulty dies can be eliminated before completing the entire test flow.

In this paper we discuss test scheduling in an abort-on-fail context, considering defect probabilities, which are usually derived from the statistical analysis of the production process or generated based on inductive fault analysis. We assume a hybrid built-in self-test (BIST) based approach, where the test set is composed of pseudorandom and deterministic patterns and propose a heuristic for calculating an efficient test schedule for the assumed test architecture.

The rest of this paper is organized as follows. In the next section the background and the motivation of our work are given. In Section 3 the test scheduling problem in the abort-on-fail hybrid BIST environment is formulated. In Section 4 the proposed heuristic is described and in Section 5 the experimental results are presented. The paper is concluded in Section 6.

2. Background and Motivation

As the number of cores on a chip is increasing, the amount of total test data required for testing such systems is growing rapidly. This may pose serious problems because of the cost and technological limitations (like speed and memory capacity) of the automated test equipment (ATE). One of the well-known solutions to this problem is to use BIST and to perform test pattern generation and output response compaction on the chip, by using pseudorandom patterns. Due to several reasons, like very long test sequences, and random pattern resistant faults, this approach may not always be efficient. Therefore different hybrid approaches have been proposed, where pseudorandom test patterns are complemented with deterministic test patterns, which are applied from the ATE or, in special situations, from the on-chip memory. These approaches are generally referred to as hybrid BIST [3, 4]. Such a hybrid approach reduces the memory requirements compared to the pure deterministic testing, while providing higher fault coverage and reduced test times compared to the stand-alone BIST solution.

2.1. Hybrid BIST Architecture

In this paper we assume the following test architecture: Every core has its own dedicated BIST logic that is capable to produce a set of independent pseudorandom test patterns, i.e. the pseudorandom tests for all cores can be carried out simultaneously. The deterministic tests, on the other hand, are carried out for one core at a time, which means that only one test access bus at the system level is needed. An example of a multi-core system, with such a test architecture is given in Figure 1. By using a hybrid BIST optimization methodology, the optimal combination between pseudorandom and deterministic test patterns for every individual core, for any arbitrary memory constraint, can be found [4].

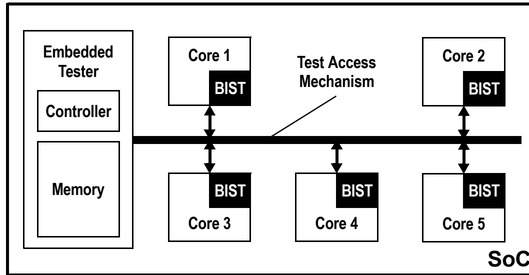

Figure 1. A hybrid BIST based SoC test architecture

By using a straightforward bus scheduling approach, a test schedule, illustrated in Figure 2, can be obtained.

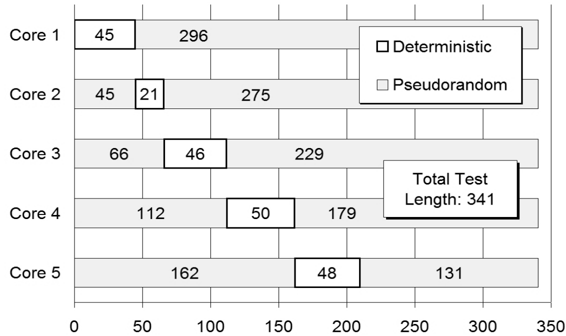

Figure 2. An example of a hybrid BIST schedule

2.2. Abort-on-Fail Test Scheduling

The SoC test scheduling problem has been studied recently by many authors [5-8]. In most of these approaches the assumption is made that all tests are applied to completion. In a production environment, however, the test can be aborted as soon as a fault has been detected. Therefore the likelihood of a block to fail during the test should be considered during test scheduling in order to improve its efficiency [9-13].

In this paper an abort-on-fail test scheduling approach in a hybrid BIST environment is proposed. A hybrid BIST approach gives us an opportunity not only to schedule the tests as fixed sequences of patterns, as it has been done in previous approaches, but also to select the best internal structure for individual test sets. This means, choosing the best order of pseudorandom and deterministic test patterns for every individual core, so that the total test time of the entire system can be minimized. During the scheduling process we will take into account the defect probabilities of individual cores and the impact of this information to the expected total test time (ETTT). We have assumed here a test-per-clock approach, although the methodology can easily be extended also for test-per-scan architectures.

3. Definitions and Problem Formulation

Suppose that a system S, consisting of cores $C_1, C_2, \ldots, C_n$, has a test architecture as depicted in Figure 1. Let us denote with DT_{ij} and PR_{ij}, respectively, the j-th deterministic pattern and j-th pseudorandom pattern in the respective sequences for core C_i. PS_{0i} and PS_{i0} denote the pseudorandom subsequences generated, respectively, before and after the deterministic sequence for core C_i. d_i and r_i are the total number of deterministic and pseudorandom test patterns for core C_i and r_{0i} and r_{i0} are the numbers of pseudorandom test patterns in the subsequence PS_{0i} and PS_{i0}, respectively.

In this paper we consider that from the test scheduling point of view the deterministic test sequence is an undividable block, i.e. the deterministic test sequence cannot be divided into smaller subsequences. The pseudorandom test sequence, on the other hand, can be stopped and restarted in order to enable the application of the deterministic test sequence at a certain scheduled time moment. This assumption is a consequence of the assumed test architecture, where pseudorandom patterns are generated locally by every core, while deterministic test patterns are applied from one single test source and applied to one core at a time. This leads us to a problem where also the test scheduling becomes hybrid: the pseudorandom test sequences can be scheduled concurrently, while the deterministic test sequences can be scheduled sequentially.

3.1. System Defect Probability

The system defect probability $DF(S)$ is defined as the probability of a system to be detected faulty during the production test. Similarly, the core defect probability $DF(C_i)$ is defined as the probability of a core to be detected faulty during the production test. Given the core defect probabilities, the system defect probability can be calculated as follows:

$$DF(S) = 1 - \prod_{i=1}^{n} (1 - DF(C_i)) \ . \quad (1)$$

In order to define the expected total test time, the individual fault coverage of a test pattern, the passing probability of a test pattern, and the accumulative passing probability of a pseudorandom subsequence have to be defined.

3.2. Individual Fault Coverage of a Test Pattern

To test a core C_i we have to apply d_i deterministic test patterns and r_i pseudorandom test patterns (in particular cases

one of these sets may be empty). By calculating the number of faults $F(DT_{ij})$ or $F(PR_{ij})$ that are not yet detected[2] by the previous patterns before the j-th pattern and detected just by the j-th pattern (either pseudorandom or deterministic), the individual fault coverage of each test pattern can be calculated as follows:

$$FC(DT_{ij}) = \frac{F(DT_{ij})}{F_i}, (1 \le j \le d_i) \ , \ FC(PR_{ij}) = \frac{F(PR_{ij})}{F_i}, (1 \le j \le r_i) \ , \qquad (2)$$

where $FC(DT_{ij})$ or $FC(PR_{ij})$ are respectively the individual fault coverage of the j-th deterministic and j-th pseudorandom pattern for core C_i. F_i is the total number of non-redundant faults in core C_i.

3.3. Passing Probability of Test Patterns

The passing probability of a test pattern is defined as the probability that the test pattern does not detect any fault. This probability can be calculated from the defect probability of the core and from the individual fault coverage of the test pattern:

$$P(DT_{ij}) = 1 - FC(DT_{ij}) \times DF(C_i), (1 \le j \le d_i) \ . \qquad (3)$$

$$P(PR_{ij}) = 1 - FC(PR_{ij}) \times DF(C_i), (1 \le j \le r_i) \ . \qquad (4)$$

Here $P(DT_{ij})$ and $P(PR_{ij})$ are the passing probabilities of the j-th deterministic and j-th pseudorandom pattern for the core C_i, respectively. We have assumed here that the modelled faults cover the defects in a uniform way, and we have 100% defect coverage, in order to simplify the problem. These assumptions do not have a large impact on the scheduling results. In the following we say that a test has passed, when it does not detect any fault and it has failed, when it detects at least one fault.

3.4. Accumulative Passing Probability of Pseudo-random Subsequences

When considering an abort-on-fail approach, the test should be terminated as soon as a fault is detected. This can easily be achieved with deterministic test, where test responses can be analyzed after every individual test pattern and testing can be terminated at any time moment. In case of pseudorandom test sequences the test responses are usually compacted into a signature, and the test response can only be analyzed at the end of the entire pseudorandom sequence. Therefore test abortion in the case of pseudorandom patterns can occur only either at the switching moment from the pseudorandom test sequence to the deterministic test sequence or at the end of the individual pseudorandom test sequence, even if the fault occurs earlier.

In this paper we assume that pseudorandom sequences are generated locally, in isolation, and therefore the pseudorandom test of a core C_i can be executed independently from the rest of the system. We assume that the pseudorandom test of a core C_i can be executed at any moment when the deterministic test patterns are not applied to C_i. Therefore the original pseudorandom test sequence PS_i of core C_i may be divided into two subsequences PS_{0i} and PS_{i0}, denoting the pseudorandom test sequence applied before and after the deterministic test sequence, correspondingly. Both having their accumulative passing probability:

$$P(PS_{0i}) = \prod_{j=1}^{r_{0i}} P(PR_{ij}) \ , \ P(PS_{i0}) = \prod_{j=r_{0i}+1}^{r_{0i}+r_{i0}} P(PR_{ij}) \ . \qquad (5)$$

3.5. Expected Total Test Time

As illustrated in Figure 3 the entire test procedure can be divided into two sessions. During the first session the deterministic patterns are applied sequentially to all cores. At the same time up to n-1 pseudorandom sequences are also generated and applied to their respective cores. In this session the test can be terminated as soon as a fault is detected by a deterministic test pattern (*DT*) or at the end of any pseudorandom sequence (*PS*), if a fault was detected by the pseudorandom sequence. The second session contains only concurrently generated pseudorandom test sequences and, consequently, the test can be terminated only at the end of any of those sequences. In a hybrid BIST environment the test length for all cores is usually the same (see Figure 2). In this example we have generalized the problem and consider also the situation where different cores have different test lengths, which is also assumed in our algorithm.

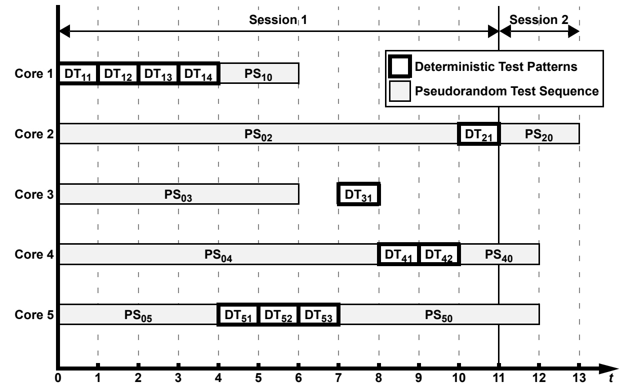

Figure 3. Hybrid BIST sessions for a system with 5 cores

Since every deterministic test pattern as well as pseudorandom test sequence has its own passing probability, an accumulative probability of test termination for any time moment together with an expected total test time of the entire system can be calculated.

Let us assume that we have a hybrid test sequence which terminates at time point t_a and there are totally m possible test termination points t_1, t_2, ... , t_m. Based on the previously presented formulas the probability of a test to fail, $p(t_k)$, can be calculated for any possible test termination point t_k ($1 \le k \le m$). Similarly the probability that the entire test sequence passes, without detecting a fault, $q(t_a)$, can be calculated. The expected total test time (ETTT) of a system S can be defined as follows:

$$ETTT = \sum_{k=1}^{m} p(t_k) T(t_k) + q(t_a) T(t_a) \ , \qquad (6)$$

2. This information can be obtained from fault simulation

where $T(t_k)$ and $T(t_a)$ denote the time from the beginning of the test to the test termination moments, indicated by t_k and t_a, respectively.

As outlined earlier in the section, the entire test process can be divided into two test sessions, where the test termination can take place either after every individual pattern (Session 1) or after every pseudorandom sequence (Session 2). We have:

$$ETTT = \sum_{i=1}^{n}\sum_{j=1}^{d_i} p(DT_{ij})T(DT_{ij}) + \sum_{l=1}^{r} p(PS_l)T(PS_l) + q(t_a)T(t_a)\,, \qquad (7)$$

where DT_{ij} denotes the j-th deterministic test pattern of a core C_i, and PS_l denotes the l-th pseudorandom test sequence in session two. $T(DT_{ij})$ denotes the time from the beginning of the test up to and including pattern DT_{ij} and $p(DT_{ij})$ is a probability that faults are detected by the pattern DT_{ij}. Similarly $p(PS_l)$ is the probability that faults are detected by the sequence PS_l and $T(PS_l)$ is the time from the beginning of the test up to and including sequence PS_l. $q(t_a)$ and $T(t_a)$ have the same meaning as in Equation (6).

At any test termination point the test sequences can be divided into 2 sets. Set S_p contains all the sequences (pseudorandom and deterministic) that have been applied and passed without detecting a fault, while set S_f contains deterministic patterns and pseudorandom sequences that detected the fault concurrently. By adding all the products of the test time and fault detection probabilities at every possible termination point, we have finally:

$$\begin{aligned} ETTT = &\sum_{i=1}^{n}\sum_{j=1}^{d_i}\left(\prod_{\forall x \in S_p(DT_{ij})} P(x)\right)\left(1-\prod_{\forall y \in S_f(DT_{ij})} P(y)\right)T(DT_{ij}) + \\ &\sum_{l=1}^{r}\left(\prod_{\forall x \in S_p(PS_l)} P(x)\right)\left(1-\prod_{\forall y \in S_f(PS_l)} P(y)\right)T(PS_l) + \left(\prod_{\forall x \in S_p(t_a)} P(x)\right)T(t_a) \end{aligned} \qquad (8)$$

In Figure 4 we have illustrated graphically the situation, when a test is aborted before completion. In this example we have assumed, that the fault has been detected by the pseudorandom sequence PS_{10} and the test response has been analyzed at the end of the sequence at time moment t=6. At the same time moment also the deterministic test pattern DT_{52} was applied and the pseudorandom sequence PS_{03} finished. Both of them, DT_{52} and PS_{03}, may have detected a fault as well. In this situation $S_p(PS_{10}) = \{DT_{11}, DT_{12}, DT_{13}, DT_{14}, PS_{05}, DT_{51}\}$ contains all tests that have been successfully passed and $S_f(PS_{10}) = \{PS_{10}, PS_{03}, DT_{52}\}$ contains all tests that potentially detected a fault. Note that PS_{02} and PS_{04} cannot be included to neither $S_p(PS_{10})$ nor $S_f(PS_{10})$ since the results of these pseudorandom test sequences can be analyzed only at the end of the sequence. Thus the probability that the test is terminated at t=6 is:

$$\left(\prod_{\forall x \in S_p(PS_{10})} P(x)\right)\left(1-\prod_{\forall y \in S_f(PS_{10})} P(y)\right).$$

Equation (8) is essentially the sum of terms, each corresponding to a possible termination time. Each term equals the test length multiplied by the probability that the test is terminated at that point.

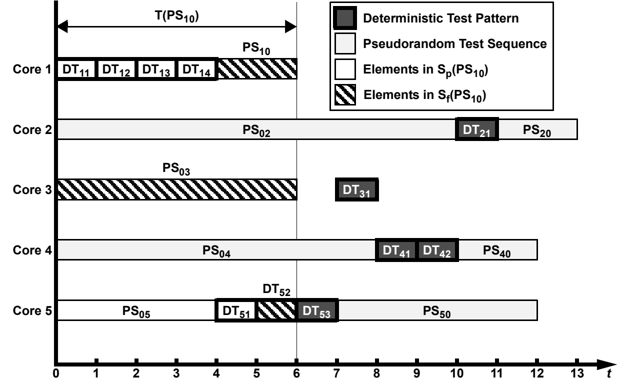

Figure 4. Example of the test termination in session 1

3.6. Problem Formulation

Let us assume a core-based system with the given test architecture as described in section 2.1. For such a system an optimal hybrid test set and a straightforward schedule can be calculated [4]. The objective of this paper is to take into account defect probabilities of individual cores and to use this information to schedule the tests in such a manner that the expected total test time (ETTT) is minimized.

4. Proposed Heuristic for Test Scheduling

Our objective is to develop an iterative heuristic for ETTT minimization. As described earlier, the test scheduling problem in the hybrid abort-on-fail test environment is essentially scheduling of deterministic test sequences, such that the ETTT of the system is minimal.

By changing the schedule of deterministic sequences, the set of passed test sequences S_p and the set of failed test sequences S_f, affiliated to every possible test termination moment, is also changed. Consequently the individual fault coverage of each test pattern must be recalculated, since the passing probability of these patterns is changed. This will lead to the recalculation of the ETTT as described in the previous section.

It would be natural to order the tests in such a way, that the cores with high failure probability would be tested first. However such a naive schedule does not necessarily lead to the minimal expected total test time. In addition to the defect probabilities also the efficiency of test patterns and the length of individual test sequences have to be taken into account. Due to the complexity of the problem we propose here an iterative heuristic that can be efficiently used.

In our heuristic we assume that we start from a subset of m already scheduled deterministic sequences, $m<n$. The objective is to increase this subset to m+1 scheduled deterministic sequences. This is obtained by selecting a deterministic sequence from the remaining unscheduled n-m candidate sequences and inserting it into the existing sequence in such a way, that the resulting ETTT is as short as possible.

This procedure is repeated for all cores $m = 0, 1, \ldots, n\text{-}1$. For the initial solution (m=0) the test sequence with the lowest ETTT is chosen.

At every iteration, $(n\text{-}m)(m+1)$ different solutions have to be explored since there are $n\text{-}m$ candidate sequences and m+1 insertion points for each candidate. The heuristic is illustrated in Figure 5. Here we have illustrated a situation where two deterministic test sequences out of five are already scheduled (m=2, n=5). For every candidate schedule there are three different insertion points, indicated by arrows. During each iteration step the ETTT for all candidate sequences for all possible insertion points is calculated and the candidate sequence will be finally inserted to the point which produces lowest ETTT.

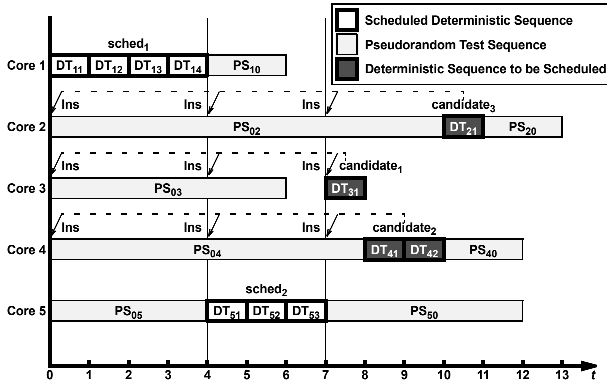

Figure 5. Initial solution for the iteration

The new situation is illustrated in Figure 6. In this example the deterministic test sequence of core 4 was chosen and inserted into the schedule after the core 1.

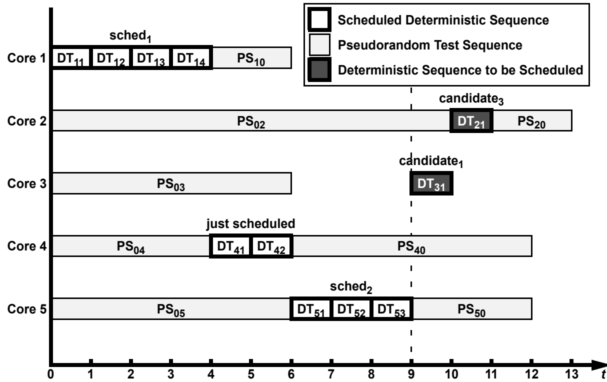

Figure 6. The new locally optimal order after the iteration

In Figure 7 the pseudo-code of the algorithm is presented. As described earlier, the test schedule is obtained constructively by enlarging the subset of scheduled test sequences. During each iteration we add one additional sequence into the previously scheduled set of sequences and explore the solution space in order to find the optimal insertion point for the new sequence. The algorithm starts with the initialization phase. Thereafter comes the iterative scheduling heuristic, consisting of three main loops. The outer loop iteratively increases the number of already scheduled sequences. The middle loop iteratively goes through all candidate sequences in the current candidate set and the inner loop calculates for the chosen candidate sequence a new ETTT in every possible insertion point. The solution with lowest ETTT is chosen and the new schedule forms an initial solution for the next iteration. The heuristic stops when all tests have been scheduled.

```
// Initialization
Scheduled[1..n] = { };
Candidate[1..n] = {1, 2, ... , n};
Min_Cost = a large value;
Candidate#_Chosen = 0;
InsertPosition#_Chosen = 0;
// Iteration
// Outer loop
for (Curr_Scheduled# = 1 to n) do {
    Min_Cost = a large value;
    m = Curr_Scheduled_Number(Scheduled[ ]);
    // Middle loop
    for (Curr_Candidate# = 1 to n-m) do
        // Inner loop
        for (Curr_InsertPosition# = 0 to m) do {
            New_Cost = Cost(Curr_Candidate#, Curr_InsertPosition#);
            if (New_Cost < Min_Cost) {
                Min_Cost = Current_Cost;
                Candidate#_Chosen = Curr_Candidate#;
                InsertPosition#_Chosen = Curr_InsertPosition#;
            }
    }
    Insert(Candidate#_Chosen, InsertPosition#_Chosen, Scheduled[ ]);
}
// Output
Output(Scheduled[1..n]);
```

Figure 7. Pseudo-code of the algorithm

The algorithm proposed here has a polynomial computational complexity of $O(kn^4)$, where n is the number of cores and k is the average number of deterministic test patterns prepared for every core.

5. Experimental Results

We have performed experiments with 9 different designs, consisting of 5 to 50 cores (Table 1). In order to obtain diversification we have calculated for every experimental design 5 different hybrid test sets (different ratio of pseudorandom and deterministic test patterns) and the experimental results illustrate the average of five experiments. The defect probabilities for individual cores have been given randomly, while keeping the system defect probability at the value 0.6.

In order to illustrate the significance of test scheduling we have performed another set of experiments for comparison, where a random schedule is assumed. As it can be seen from Table 1, by employing our heuristic the ETTT can be reduced in a range of 5-15%, which is very relevant for large volume production testing.

As our heuristic can produce only a near optimal solution, experiments for estimating the accuracy of the solution were performed. For this purpose a simulated annealing algorithm and an exhaustive search has been used, where possible. As it can be seen from Table 1 our heuristic is able to produce results similar or very close to the results obtained with simulated annealing and exhaustive search, while having significantly lower computation times. These comparisons are

also illustrated in Figure 8 and Figure 9. In Figure 8 we have compared the ETTT values, calculated with different approaches, while in Figure 9 the CPU times with our heuristic and with simulated annealing are compared.

6. Conclusions

In this paper a test scheduling methodology in the context of a hybrid BIST environment was presented. Different from other approaches the defect probabilities of individual cores were taken into consideration and a methodology for expected total test time calculation in the abort-on-fail environment was proposed. We have also developed a scheduling heuristic for expected total test time minimization, that produces near-optimal solutions with low computational overhead. Experimental results have shown the efficiency of the proposed method.

References

[1] B. T. Murray, and J. P. Hayes, "Testing ICs: Getting to the core of the problem", *IEEE Transactions on Computer*, Vol. 29, No. 11, 1996, pp. 32-38.

[2] Y. Zorian, E. J. Marinissen, and S. Dey, "Testing Embedded Core-Based System Chips", *IEEE International Test Conference (ITC)*, 1998, pp. 130-143.

[3] G. Jervan, Z. Peng, and R. Ubar, "Test Cost Minimization for Hybrid BIST", *IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT)*, 2000, pp. 283-291.

[4] G. Jervan, P. Eles, Z. Peng, R. Ubar, and M. Jenihhin, "Test Time Minimization for Hybrid BIST of Core-Based Systems", *IEEE Asian Test Symposium (ATS)*, 2003, pp. 318-323.

[5] E. Cota, L. Carro, M. Lubaszewski, and A. Orailoglu, "Test Planning and Design Space Exploration in a Core-based Environment", *Design, Automation and Test in Europe Conference (DATE),* 2002*,* pp. 478-485.

[6] 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", *IEEE Asian Test Symposium (ATS)*, 2001, pp. 265-270.

[7] V. Iyengar, K. Chakrabarty, and E. J. Marinissen, "Test Wrapper and Test Access Mechanism Co-optimization for System-on-Chip", *International Test Conference (ITC)*, 2001, pp. 1023-1032.

[8] E. Larsson, and Z. Peng, "An Integrated Framework for the Design and Optimization of SOC Test Solutions", *Journal of Electronic Testing; Theory and Applications (JETTA)*, Vol. 18, No. 4/5, 2002, pp. 385-400.

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

[10] 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, 1994, pp. 796-813.

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

[12] S. Koranne, "On Test Scheduling for Core-based SOCs", *International Conference on VLSI Design (VLSID)*, 2002, pp. 505-510.

[13] E. Larsson, J. Pouget, and Z. Peng, "Defect-Aware SOC Test Scheduling", *IEEE VLSI Test Symposium (VTS)*, 2004, pp. 359-364.

Table 1. Experimental Results

Number of Cores	5		7		10		12		15		17		20		30		50	
	ETTT	CPU Time(s)	ETTT	CPU Time(s)	ETTT	CPU Time(s)	ETTT	CPU Time(s)	ETTT	CPU Time(s)	ETTT	CPU Time(s)	ETTT	CPU Time(s)	ETTT	CPU Time(s)	ETTT	CPU Time(s)
Random Scheduling	248.97	1.1	261.38	64.4	366.39	311.8	415.89	346.8	427.34	371.6	544.37	466.6	566.13	555.4	782.88	822.4	1369.54	1378.0
Our Heuristic	228.85	0.6	232.04	1.4	312.13	6.6	353.02	12.2	383.40	25.2	494.57	43.6	517.02	85.4	738.74	380.4	1326.40	3185.0
Simulated Annealing	228.70	1144.2	231.51	1278.5	311.68	3727.6	352.10	4266.8	381.46	5109.2	493.93	6323.8	516.89	7504.4	736.51	11642.4	1324.44	21308.8
Exhaustive Search	228.70	1.2	231.51	80.0	311.68	112592.6	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A

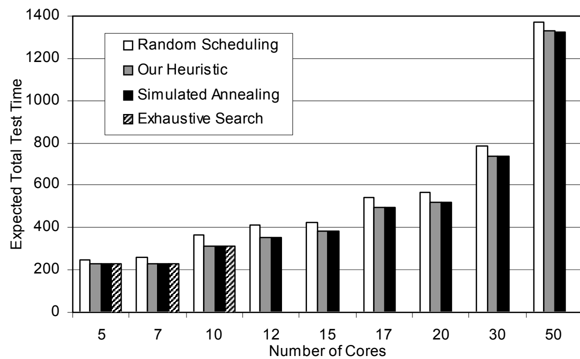

Figure 8. Comparison of expected total test times

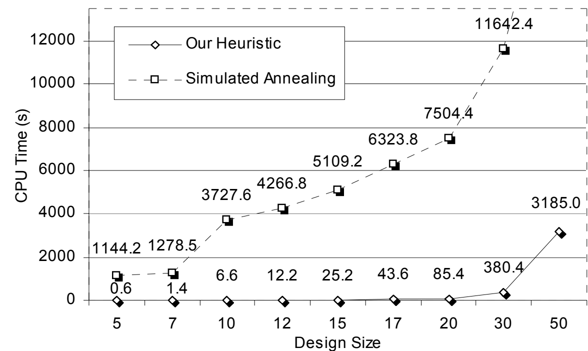

Figure 9. Comparison of CPU times