A Hybrid BIST Architecture and its Optimization for SoC Testing

Gert Jervan, Zebo Peng
Linköping University, Sweden
{gerje, zebpe}@ida.liu.se

Raimund Ubar, Helena Kruus
Tallinn Technical University, Estonia
raiub@pld.ttu.ee, helen.kruus@ttu.ee

Abstract

This paper presents a hybrid BIST architecture and methods for optimizing it to test systems-on-chip in a cost effective way. The proposed self-test architecture can be implemented either only in software or by using some test related hardware. In our approach we combine pseudorandom test patterns with stored deterministic test patterns to perform core test with minimum time and memory, without losing test quality. We propose two algorithms to calculate the cost of the test process. To speed up the optimization procedure, a Tabu search based method is employed for finding the global cost minimum. Experimental results have demonstrated the feasibility and efficiency of the approach and the significant decreases in overall test cost.

1. Introduction

The rapid advances of the microelectronics technology in recent years have brought new possibilities to integrated circuits (ICs) design and manufacturing. Many systems are nowadays designed by embedding predesigned and preverified complex functional blocks, usually referred as cores, into one single die (Figure 1). Such a design style allows designers to reuse previous designs and will lead therefore to shorter time to market and reduced cost. Such a System-on-Chip (SoC) approach is very attractive from the

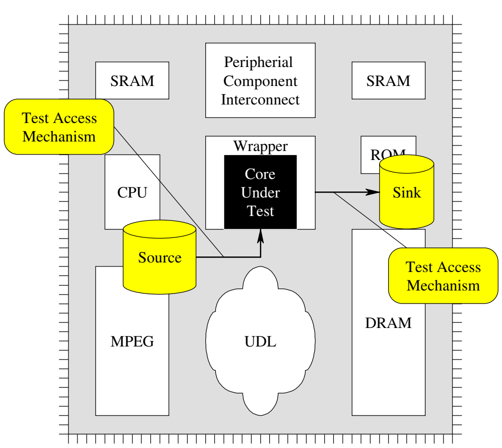

Figure 1. Testing a system-on-chip

designers' perspective. Testing of SoC, on the other hand, shares all the problems related to testing modern deep submicron chips, and introduces also some additional challenges due to the protection of intellectual property as well as the increased complexity and higher density [1].

To test the individual cores of the system the test pattern source and sink have to be available together with an appropriate test access mechanism (TAM) [2] as depicted in Figure 1. A traditional approach implements both source and sink off-chip and requires therefore the use of external Automatic Test Equipment (ATE). But, as the requirements for the ATE speed and memory size are continuously increasing, the ATE solution can be unacceptably expensive and inaccurate. Therefore, in order to apply at-speed tests and to keep the test costs under control, on-chip test solutions are becoming more and more popular. Such a solution is usually referred to as built-in self-test (BIST).

A typical BIST architecture consists of a test pattern generator (TPG), a test response analyzer (TRA) and a BIST control unit (BCU), all implemented on the chip. This approach allows at-speed tests and eliminates the need for an external tester. It can be used not only for manufacturing test but also for periodical field maintenance tests.

The classical way to implement the TPG for logic BIST (LBIST) is to use linear feedback shift registers (LFSR). But as the test patterns generated by the LFSR are pseudorandom by their nature [3], the LFSR-based approach often does not guarantee a sufficiently high fault coverage (especially in the case of large and complex designs) and demands very long test application times in addition to high area overheads. Therefore, several proposals have been made to combine pseudorandom test patterns, generated by LFSRs, with deterministic patterns [4-8], to form a hybrid BIST solution.

The main concern of the hybrid BIST approaches has been to improve the fault coverage by mixing pseudorandom vectors with deterministic ones, while the issue of cost minimization has not been addressed directly.

To reduce the hardware overhead in the LBIST architectures the hardware LFSR implementation can be replaced by software, which is especially attractive to test SoCs, because of the availability of computing resources directly in the system (a typical SoC usually contains at least one processor core). On the other hand, the software based approach, is criticized because of the large memory requirements (to store the test program and test patterns). Similar work has been reported in [7]. However, the approach presented there has no direct cost considerations and can therefore lead to very long test application times because of the unlimited number of pseudorandom test patterns.

In our approach we propose to use a hybrid test set, which contains a limited number of pseudorandom and deterministic test vectors. The pseudorandom test vectors can be generated either by hardware or by software and later complemented by the stored deterministic test set which is specially designed to shorten the pseudorandom test cycle and to target the random resistant faults. The basic idea of Hybrid BIST was discussed in [13].

The main objective of the current work is to propose a test architecture that supports the combination of pseudorandom and deterministic vectors and to find the optimal balance between those two test sets with minimum cost of time and memory, without losing test quality. We propose two different algorithms to calculate the total cost of the hybrid BIST solution and a fast method to find the optimal switching moment from the pseudorandom test to the stored deterministic test patterns.

A similar problem has been addressed in [8], where an approach to minimize testing time has been presented. It has shown that hybrid BIST (or CBET in their terminology) can achieve shorter testing time than pseudorandom or deterministic test alone. However, the proposed algorithm does not address total cost minimization (time and memory).

The rest of this paper is organized as follows. In section 2 we introduce the target architecture to implement our approach, section 3 gives an overview of the concepts of hybrid BIST. In sections 4 and 5 we discuss the concepts of calculating the test cost for different solutions, including hybrid BIST. In section 6 Tabu search based method is proposed for optimizing the hybrid BIST test set and in section 7 we present the experimental results which demonstrate the efficiency of our approach. In section 8 we will draw some conclusions together with an introduction to future work.

2. Target Hybrid BIST Architecture

A hardware based hybrid BIST architecture is depicted in Figure 2, where the pseudorandom pattern generator (PRPG) and the Multiple Input Signature Analyzer (MISR) are implemented inside the core under test (CUT). The deterministic test pattern are precomputed off-line and stored inside the system.

To avoid the hardware overhead caused by the PRPG and MISR, and the performance degradation due to excessively large LFSRs, a software based hybrid BIST can be used where pseudorandom test patterns are produced by the test software. However, the cost calculation and optimization algorithms to be proposed are general, and can be applied as well to the hardware based as to the software based hybrid BIST optimization.

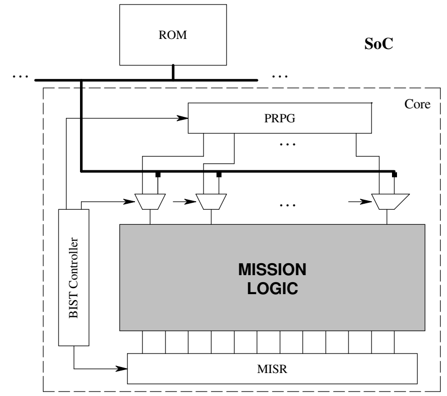

Figure 2. Hardware based hybrid BIST architecture

In case of software based solution, the test program, together with test data (LFSR polynomials, initial states, pseudorandom test length, signatures), is kept in a ROM. The deterministic test vectors are generated during the development process and are stored in the same place. For transporting the test patterns, we assume that some form of TAM is available.

In test mode the test program is executed in the processor core. The test program proceeds in two successive stages. In the first stage the pseudorandom test pattern generator, which emulates the LFSR, is executed. In the second stage the test program will apply precomputed deterministic test vectors to the core under test.

The pseudorandom TPG software is the same for all cores in the system and is stored as one single copy. All characteristics of the LFSR needed for emulation are specific to each core and are stored in the ROM. They will be loaded upon request. Such an approach is very effective in the case of multiple cores, because for each additional core, only the BIST characteristics for this core have to be stored. The general concept of the software based pseudorandom TPG is depicted in Figure 3.

Although it is assumed that the best possible pseudorandom sequence is used, not always all parts of the system are testable by a pure pseudorandom test sequence. It can also take a very long test application time to reach a

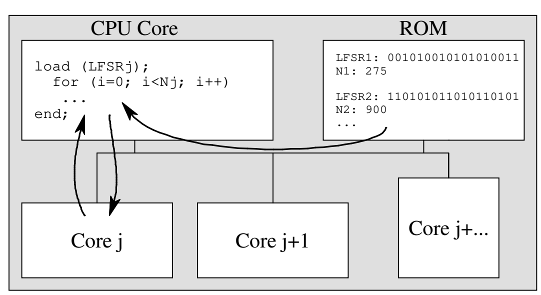

Figure 3. LFSR emulation

good fault coverage level.

In case of hybrid BIST, we can dramatically reduce the length of the initial pseudorandom sequence by complementing it with deterministic stored test patterns, and achieve the 100% fault coverage. The method proposed in the paper helps to find tradeoffs between the length of the best pseudorandom test sequence and the number of stored deterministic patterns.

3. Cost of Hybrid BIST

Since the test patterns generated by LFSRs are pseudorandom by nature, the generated test sequences are usually very long and not sufficient to detect all the faults. Figure 4 shows the fault coverage of the pseudorandom test as the function of the test length for some larger ISCAS'85 [9] benchmark circuits. To avoid the test quality loss due to the random pattern resistant faults and to speed up the testing process, we have to apply deterministic test patterns targeting the random resistant and difficult to test faults. Such a hybrid BIST approach starts with a pseudorandom test sequence of length L. On the next stage, stored deterministic test approach takes place: precomputed test patterns, stored in the ROM, are applied to the core under test to reach the desirable fault coverage.

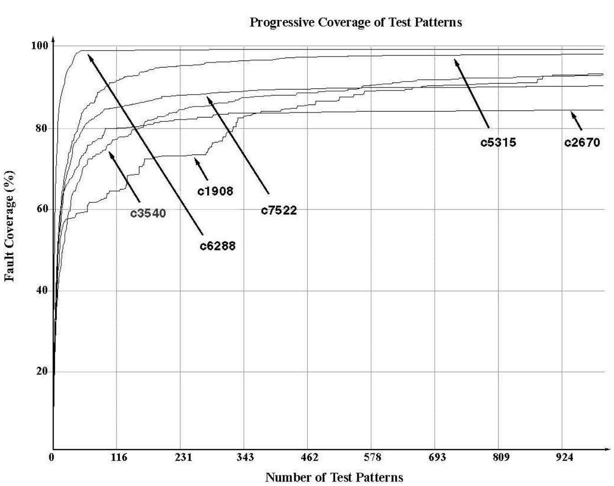

Figure 4. Pseudorandom test for some ISCAS'85 circuits

In a hybrid BIST technique the length of the pseudorandom test L is an important parameter, which determines the behavior of the whole test process [7]. It is assumed in this paper that for the hybrid BIST the best polynomial for the pseudorandom sequence generation will be chosen. Removing the latter part of the pseudorandom sequence leads to a lower fault coverage achievable by the pseudorandom test. The loss in fault coverage should be covered by additional deterministic test patterns. In other words, a shorter pseudorandom test set implies a larger deterministic test set. This requires additional memory space, but at the same time, it shortens the overall test process. A longer pseudorandom test, on the other hand, will lead to longer test application time with reduced memory requirements. Therefore it is crucial to determine the optimal length of the pseudorandom test L_{OPT} in order to minimize the total testing cost.

Figure 5 illustrates the total cost calculation for the hybrid BIST consisting of pseudorandom test and stored test, generated off-line. We can define the total test cost of the hybrid BIST C_{TOTAL} as:

$$C_{TOTAL} = C_{GEN} + C_{MEM} = \alpha L + \beta S \qquad (1)$$

where C_{GEN} is the cost related to the time for generating L pseudorandom test patterns (number of clock cycles), C_{MEM} is related to the memory cost for storing S precomputed test patterns to improve the pseudorandom test set, and α, β are constants to map the test length and memory space to the costs of the two parts of the test solutions to be mixed. Figure 5 illustrates how the cost of pseudorandom test is increasing when striving to higher fault coverage (the C_{GEN} curve). The total cost C_{TOTAL} is the sum of the above two costs. The weights α and β reflect the correlation between the cost and the pseudorandom test time (number of clock cycles used) and between the cost and the memory size needed for storing the precomputed test sequence, respectively. For simplicity we assume here $\alpha = 1$, and $\beta = B$ where B is the number of bytes of the input test vector to be applied on the CUT. Hence, to carry out some experimental work for demonstrating the feasibility and efficiency of the following algorithms, we use as the cost units the number of clocks used for pseudorandom test generation and the number of bytes in the memory needed for storing the precomputed deterministic test patterns. In practice those weights are determined by the system specification and requirements and can be used to drive the final implementation towards different alternatives (for example slower, but more memory efficient solution).

Equation 1, which is used for calculating the test cost as a sum of costs of pseudorandom test, and of the memory associated with storing the ATPG produced test, represents a

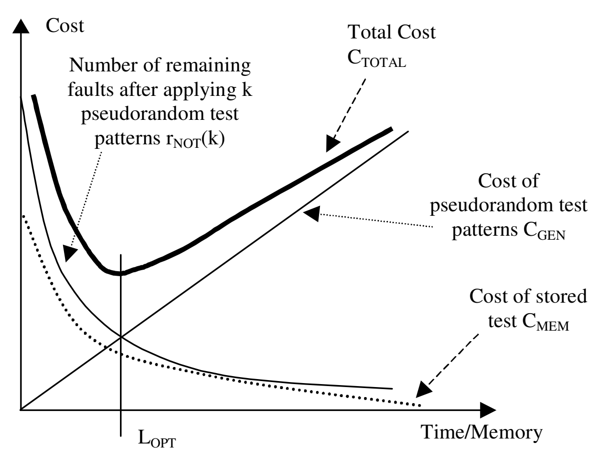

Figure 5. Cost calculation for hybrid BIST

simplified cost model for the hybrid BIST. In this model neither the basic cost of memory (or its equivalent) occupied by the LFSR emulator, nor the time needed for generating deterministic test patterns are taken into account. However, the goal of this paper was not to develop accurate cost function for the whole BIST solution. The goal was to show that the total cost of a hybrid BIST is essentially a function of arguments L and S, and to develop a method to calculate the value of S at a given value of L to find the tradeoffs between the length of pseudorandom test and the number of deterministic patterns to minimize the total cost of a hybrid BIST.

Hence, the main problem of the formulated optimization task is how to find the curves C_{GEN} and C_{MEM} in Figure 5 in a most efficient way.

4. Calculation of the Cost for Pseudorandom Test

Creating the curve $C_{GEN} = \alpha L$ is not difficult. For this purpose, the cumulative fault coverage (like in Figure 4) for the pseudorandom sequence generated by a LSFR should be calculated by a fault simulation. As the result we find for each clock cycle the list of faults which were covered at this time moment. In fact, we are interested to identify only these clock numbers at which at least one new fault will be covered. Let us call such clock numbers and the corresponding pseudorandom test patterns *efficient clocks* and *efficient patterns*, respectively.

As an example, in Table 1 the first four columns represent a fragment of selected results of fault simulation for pseudorandom test in the case of the ISCAS'85 circuit c880, where

- k *is the number of the clock cycle,*
- $r_{DET}(k)$ is the number of new faults detected by the test pattern generated at the clock signal k,
- $r_{NOT}(k)$ is the number of faults not yet covered with the sequence generated by k clock signals,
- $FC(k)$ is the fault coverage reached with the sequence generated by k clock signals.

Table 1. BIST analysis data

k	$r_{DET}(k)$	$r_{NOT}(k)$	$FC(k)$	$t(k)$
1	155	839	15.6%	104
2	76	763	23.2%	104
3	65	698	29.8%	100
4	90	608	38.8%	101
5	44	564	43.3%	99
10	104	421	57.6%	95
20	44	311	68.7%	87
50	51	218	78.1%	74
100	16	145	85.4%	52
200	18	114	88.5%	41
411	31	70	93.0%	26
954	18	28	97.2%	12
1560	8	16	98.4%	7
2153	11	5	99.5%	3
3449	2	3	99.7%	2
4519	2	1	99.9%	1
4520	1	0	100.0%	0

The rows in Table 1 correspond to selected efficient clocks for the circuit c880. If we decide to switch from pseudorandom mode to the deterministic mode after the clock number k, then $L = k$.

More difficult is to find the values for $C_{MEM} = \beta S$. Let $t(k)$ be the number of test patterns needed to cover $r_{NOT}(k)$ not yet detected faults (these patterns should be precomputed and used as stored test patterns in the hybrid BIST). As an example, this data for the circuit c880 are depicted in the last column of Table 1. In the following section the difficulties and possible ways to solve the problem are discussed.

5. Calculation of the Cost for Stored Test

There are two approaches to find $t(k)$: ATPG based and fault table based. Let us have the following notations:

- i – *the current number of the entry in the table of BIST analysis data;*
- k(i) – *the number of the efficient clock cycle;*
- $R_{DET}(i)$ - *the set of new faults detected by the pseudorandom pattern generated at* k(i)*;*
- $R_{NOT}(i)$ - *the set of not yet covered faults after applying the pseudorandom pattern number* k(i)*;*
- T(i) - *the set of test patterns found by ATPG to cover the faults in* $R_{NOT}(i)$*;*
- N – *the number of all efficient patterns in the sequence created by the pseudorandom test;*
- FT – *the fault table for a given set of test patterns* T *and for the given set of faults* R*: the table defines the subsets* $R(t_j) \in R$ *of detected faults for each pattern* $t_j \in T$.

Algorithm 1: ATPG based generation of *t(k)*

1. Let $k := N$;
2. Generate for $R_{NOT}(k)$ a test $T(k)$, $T := T(k)$, $t(k) := |T|$;
3. For all $k = N-1, N-2, \ldots 1$:
   Generate for the faults $R_{NOT}(k)$ not covered by T a test set $T(k)$, $T := T + T(k)$, $t(k) := |T|$;
4. END.

This algorithm generates a new deterministic test set for the not yet detected faults at every efficient clock cycle. In this way we have the complete test set (consisting of pseudorandom and deterministic test vectors) for every efficient clock, which can reach to the maximal achievable fault coverage. The number of deterministic test vectors at all efficient clocks are then used to create the curve $C_{MEM}(\beta S)$. The algorithm is straightforward, however, very time consuming because of repetitive use of ATPG.

Algorithm 2: Fault Table based generation of *t(k)*

1. Calculate the whole test $T = \{t_j\}$ for the whole set of faults R by any ATPG to reach as high fault coverage as possible;
2. Create for T and R the fault table $FT = \{R(t_j)\}$;
3. Take $k = 0$, $T_k = T$, $R_k = R$, $FT_k = FT$;
4. Take $k = k + 1$;

5. Calculate by fault simulation $R_{DET}(k)$;
6. Update the fault table: $\forall j$, $t_j \in T_k$: $R(t_j) - R_{DET}(k)$;
7. Remove from the test set T_k all the test patterns $t_j \in T_k$ where $R(t_j) = \emptyset$;
8. If $T(k) = \emptyset$ go to END;
9. Optimize the test set T_k by any test compaction algorithm; $t(k) = | T_k |$; go to 4;
10. END.

This algorithm starts by generating a test set T for all detectable faults. Based on the fault simulation results a fault table FT will be created. By applying k pseudorandom patterns, we can remove from the original fault table all faults, which were covered by the pseudorandom vectors and by using static test compaction reduce the original deterministic test set. Those modifications should be performed iteratively for all possible breakpoints to calculate the curve $C_{MEM}(\beta S)$ and to use this information to find the optimal C_{TOTAL}.

More details about the algorithm can be found in [10], but in the case of very large circuits both of these algorithms may lead to very expensive and time-consuming experiments. It would be desirable to find the global optimum of the total cost curve by as few sampled calculations of the total cost for selected values of k as possible.

6. Tabu search

For reducing the number of Total Cost calculations in Algorithms 1 and 2 for finding the minimum value, the method of Tabu search [11-12] as a general iterative heuristic for solving combinatorial optimization problems was used.

Algorithm 3: Tabu search
Start with initial solution $SO \subset \Omega$;
BestSolution:=SO;
$T:=\emptyset$;
While number of empty iterations < E **Or** there is no return to previously visited solution **Do**
    Generate the sample of neighbor solutions $V^* \subset N(SO)$;
    Find best $Cost(SO^* \subset V^*)$;
M: **If** move to solution SO^* is not in the T **Then**
        $SO^{trial} := SO^*$;
        Update Tabu list;
    Else
        Find the next best $Cost(SO^* \subset V^*)$;
        Go to M;
    End If;
    If $Cost(SO^{trial}) < Cost(BestSolution)$ **Then**
        $BestSolution := SO^{trial}$;
    Else
        Increment number of empty iterations E;
    End If;
End While;
END.

Tabu search is a form of local neighborhood search. Each solution $SO \in \Omega$ where Ω is the search space (the set of all feasible solutions) has an associated set of neighbors $N(SO) \subseteq \Omega$. A solution $SO' \in N(SO)$ can be reached from SO by an operation called a *move* to SO'. At each step, the local neighborhood of the current solution is explored and the best solution is selected as a new current solution. Unlike local search which stops when no improved new solution is found in the current neighborhood, Tabu search continues the search from the best solution in the neighborhood even if it is worse than the current solution. To prevent cycling, information pertaining to the most recently visited solutions are inserted in a list called *Tabu list*. Moves to the Tabu solutions are not allowed. The Tabu status of a solution is overridden when a certain criteria (aspiration criteria) is satisfied. One example of an aspiration criterion is when the cost of the selected solution is better than the best seen so far, which is an indication that the search is actually not cycling back, but rather moving to a new solution not encountered before [12].

The procedure of the Tabu search starts from an initial feasible solution SO (current solution) in the search space Ω. In our approach we use a fast estimation method proposed in [13] to find an initial solution. This estimation method is based on number of not yet covered faults $R_{NOT}(i)$ and can be obtained from the pseudorandom test simulation results (Table 1). A neighborhood $N(SO)$ is defined for each SO. Based on the experimental results it was concluded, that the most efficient step size for defining the neighborhood $N(SO)$ was 3% of efficient clocks, as the larger step size, even if it can give considerable speedup, will decrease the accuracy of the final result. A sample of neighbor solutions $V^* \subset N(SO)$ is generated. An extreme case is to generate the entire neighborhood, that is to take $V^* = N(SO)$. Since this is generally impractical (computationally expensive), a small sample of neighbors $(V^* \subset N(SO))$ is generated, and called *trial* solutions $(|V^*| = n << |N(SO)|)$. In case of ISCAS'85 benchmark circuits the best results were obtained, when the size of the sample of neighborhood solutions was 4. Increase of the size of V^* had no effect to the improvement of the results. From these trial solutions the best solution say $SO^* \in V^*$, is chosen for consideration as the next solution. The move to SO^* considered even if SO^* is worse than SO, that is, $Cost(SO^*) > Cost(SO)$. A move from SO to SO^* is made provided certain conditions are satisfied. The best candidate solution $SO^* \in V^*$ *may* or *may not* improve the current solution but is still considered. It is this feature that enables *escaping* from local optima.

One of the parameters of the algorithm is the size of the Tabu list. A Tabu list T is maintained to prevent returning to previously visited solutions. The list contains information that to some extent forbids the search from returning to a previously visited solutions. Generally the Tabu list size is small. The size can be determined by experimental runs, watching the occurrence of cycling when the size is too small, and the deterioration of solution quality when the size is too large [14]. Results have shown that the best average size for the ISCAS'85 benchmark family was 3. Larger size lead to the loss of result quality.

Let's have the following additional notations: E - number of allowed empty iterations (i.e. iterations that do not result in finding a new best solution), defined for each circuit, and SO^{trial} - solution generated from current solution as a result of the move.

7. Experimental results

Experiments were carried out on the ISCAS'85 benchmark circuits for comparing the algorithms 1 and 2, and for investigating the efficiency of the Tabu method for optimizing the hybrid BIST. Experiments were carried out using the Turbo Tester toolset [15] for deterministic test pattern generation, fault simulation, and test set compaction. The results are presented in Table 2 and illustrated by a diagram in Figure 6.

In the columns of Table 2 the following data is depicted: ISCAS'85 benchmark circuit name, L - length of the pseudorandom test sequence, C - fault coverage, S - number of test patterns generated by deterministic ATPG to be stored in BIST, C_T – total cost of BIST, T_G - the time (sec) needed for ATPG to generate the deterministic test set, T_A - the time (sec) needed for carrying out manipulations on fault tables (subtracting faults, and compacting the test set), N - number of efficient patterns in the pseudorandom test sequence, T_1 and T_2 - the time (sec) needed for calculating the cost curve by Algorithms 1 and 2, T_3 – the time (sec) to find the optimal cost by using Tabu search. T_S – the number of calculations in Tabu search, Acc – accuracy of the Tabu search solution in percentage compared to the exact solution found from the full cost curve, The total testing time for Algorithms 1 and 2 and for Tabu Search was calculated as follows:

$$T_1 = N * T_G$$
$$T_2 = T_G + N * T_A$$
$$T_3 = T_2 * (T_S / N) + \Delta,$$

where Δ is the time needed to perform the Tabu search calculations (was below 0.1 sec in the given experiments)

In fact, the values for T_G and T_A differ for the different

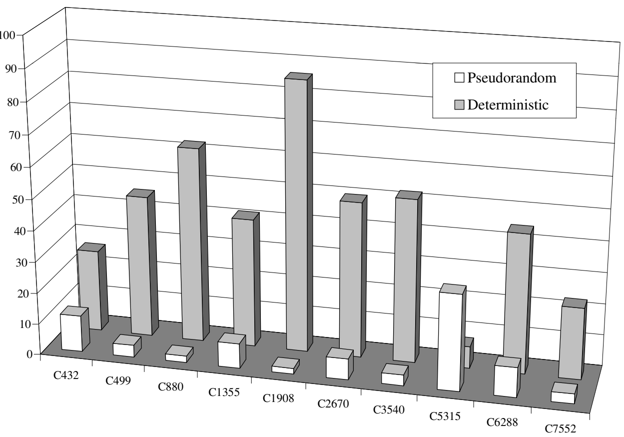

Figure 6. Percentage of test patterns in the optimized test sets compared to the original test sets

values of $i = 1,2, \ldots, N$. However the differences were in the range of few percents, which allowed us to neglect this impact and to use the average values of T_G and T_A.

In Figure 6 the amount of pseudorandom and deterministic test patterns in the optimal BIST solution is compared to the sizes of pseudorandom and deterministic test sets when only either of the sets is used. In the typical cases less than half of the deterministic vectors and only a small fraction of pseudorandom vectors are needed, however

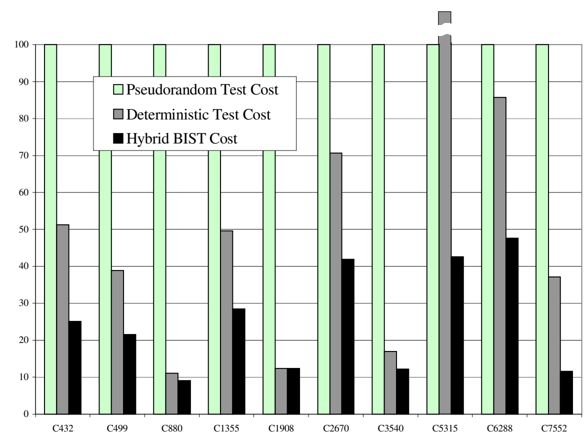

Figure 7. Cost comparison of different methods. Cost of pseudorandom test is taken as 100%

Table 2. Experimental results

Circuit	*Pseudorandom test*		*Stored test*		*Hybrid test*			*Calculation cost*							
	L	*C*	*S*	*C*	*L*	*S*	C_T	T_G	T_A	*N*	T_1	T_2	T_3	*Ts*	*Acc*
C432	780	93.0	80	93.0	91	21	196	20.1	0.01	81	1632	21	2.85	11	100.0
C499	2036	99.3	132	99.3	78	60	438	0.7	0.02	114	74	3	0.50	19	100.0
C880	5589	100.0	77	100.0	121	48	505	0.2	0.02	114	17	2	0.26	15	99.7
C1355	1522	99.5	126	99.5	121	52	433	1.2	0.03	109	133	5	0.83	18	99.5
C1908	5803	99.5	143	99.5	105	123	720	11.7	0.07	183	2132	25	3.83	28	100.0
C2670	6581	84.9	155	99.5	444	77	2754	1.9	0.09	118	230	13	0.99	9	99.1
C3540	8734	95.5	211	95.5	297	110	1067	85.3	0.14	265	22601	122	7.37	16	100.0
C5315	2318	98.9	171	98.9	711	12	987	10.3	0.11	252	2593	38	1.81	12	97.2
C6288	210	99.3	45	99.3	20	20	100	3.8	0.04	53	200	6	1.70	15	100.0
C7552	18704	93.7	267	97.1	583	61	2169	53.8	0.27	279	15004	129	3.70	8	99.7

the maximum achievable fault coverage is guaranteed and achieved. Figure 7 compares the costs of different approaches using for Hybrid BIST cost calculation an equation 1 with the parameters $\alpha = 1$, and $\beta = B$ where B is the number of bytes of the input test vector to be applied on the CUT. As pseudorandom test is usually the most expensive method, it has been selected as a reference and given value 100%. The other methods give considerable reduction in terms of cost and as it can be seen, hybrid BIST approach has significant advantage compare to the pure pseudorandom or stored test approach in most of the cases.

8. Conclusions

This paper describes a hybrid BIST architecture for testing systems-on-chip. It supports the combination of pseudorandom test patterns with deterministic test patterns in cost effective way. The self-test architecture can be implemented either in classical way, by using LFSRs, or in software to reduce the area overhead and to take advantage of the SoC architecture. For selecting the optimal switching moment from pseudorandom test mode to stored test mode two algorithms were proposed for calculating the complete cost curve of the different hybrid BIST solutions. The first one is a straightforward method based on using traditional fault simulation and test pattern generation approach. The second one is based on fault table manipulations and uses test compaction tool. The speed-up of the second algorithm was in the range from 7 to 184 (in average 64,5) for the ISCAS'85 benchmark family. A Tabu search algorithm was also developed to reduce the number of calculations in search for the optimal solution for hybrid BIST. The speed-up of using Tabu search varies from 3,5 to 34,9 (in average 10,5) compare to the faster, fault table manipulations based algorithm, whereas the accuracy of the solution (the reached minimum cost compared to the exact minimum) was not less than 97,2 % for the whole family of ISCAS'85 benchmark circuits.

The experimental results demonstrate the feasibility of the method and algorithms proposed, and the efficiency of the fault table based cost calculation method combined with Tabu search for finding optimized cost-effective solutions for hybrid BIST.

As a future work we would like to investigate possibilities to use the proposed approach for parallel testing issues (testing multiple cores simultaneously) and to use the same ideas in case of sequential cores. Additionally different search algorithms should be investigated to find the best possible method to find the optimal balance point from the search space.

Acknowledgements

This work has been supported by the EC projects IST 2000-29212 COTEST and INCO-COPERNICUS 977133 VILAB "Microelectronics Virtual Laboratory for Cooperation in Research and Knowledge Transfer", by the Estonian Science Foundation grants G3658 and 4300. The authors appreciate the work of Jaan Raik and Elmet Orasson from Tallinn Technical University for developing software tools.

References

[1] E. J. Marinissen, Y. Zorian, "Challenges in Testing Core-Based System ICs," *IEEE Communications Magazine,* pp. 104-109, June 1999.

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

[3] S. W. Golomb, "Shift Register Sequences," Aegan Park Press, Laguna Hills, 1982.

[4] S. Hellebrand, S. Tarnick, J. Rajski, B. Courtois, "Generation Of Vector Patterns Through Reseeding of Multiple-Polynomial Linear Feedback Shift Registers," *IEEE Int. Test Conference (ITC'92)*, pp. 120-129, Baltimore, 1992.

[5] M. Chatterjee, D. K. Pradhan, "A novel pattern generator for near-perfect fault-coverage," *VLSI Test Symposium*, pp. 417-425, 1995

[6] N. Zacharia, J. Rajski, J. Tyzer, "Decompression of Test Data Using Variable-Length Seed LFSRs," *VLSI Test Symposium*, pp. 426-433, 1995.

[7] S. Hellebrand, H.-J. Wunderlich, A. Hertwig, "Mixed-Mode BIST Using Embedded Processors," *Journal of Electronic Testing: Theory and Applications*, pp. 127-138, No. 12, 1998

[8] M. Sugihara, H. Date, H. Yasuura, "Analysis and Minimization of Test Time in a Combined BIST and External Test Approach," *Design, Automation & Test In Europe Conference (DATE 2000)*, pp. 134-140, Paris, France, March 2000

[9] F. Brglez, H. Fujiwara, "A Neutral Netlist of 10 Combinational Benchmark Circuits and a Target Translator in Fortran," *IEEE Int. Symp. on Circuits and Systems*, pp. 663-698, June 1985.

[10] R. Ubar, G. Jervan, Z. Peng, E. Orasson, R. Raidma, "Fast Test Cost Calculation for Hybrid BIST in Digital Systems," *Euromicro Symposium on Digital Systems Design,* pp. 318-325, Sept. 2001.

[11] F. Glover and M. Laguna. "Modern Heuristic Techniques for Combinatorial Problems", Blackwell Scientic Publishing, pp. 70-141, 1993.

[12] F.Glover, E. Taillard, and D. de Werra. "A user's guide to tabu search," *Annals of Operations Research*, 41:3-28, 1993.

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

[14] S. M. Sait, H. Youssef. "Iterative Computer Algorithms with Application in Engineering. Solving Combinatorial Optimization Problems." IEEE Computer Society Press, Los Alamitos, CA, 1999.

[15] Turbo Tester Reference Manual. Version 3.99.03. Tallinn Technical University 1999. http://www.pld.ttu.ee/tt