Proc. Estonian Acad. Sci. Eng., 2006, **12**, 3-2, 300–322

# Hybrid BIST methodology for testing core-based systems

Gert Jervan[a], Raimund Ubar[a] and Zebo Peng[b]

[a] Department of Computer Engineering, Tallinn University of Technology, Raja 15, 12618 Tallinn, Estonia; {gerje, raiub}@pld.ttu.ee
[b] Embedded Systems Laboratory, Linköping University, SE-581 83 Linköping, Sweden; zpe@ida.liu.se

Received 15 May 2006, in revised form 31 July 2006

**Abstract.** This paper describes a hybrid BIST methodology for testing systems-on-chip. In our hybrid BIST approach a test set is assembled, for each core, from pseudorandom test patterns that are generated on-line, and deterministic test patterns that are generated off-line and stored in the system. The deterministic test set is specially designed to shorten the pseudorandom test cycle and to target random resistant faults. To support such a test strategy, we have developed several hybrid BIST architectures that target different test scenarios. As the test lengths of the two test sequences is one of the important parameters in the final test cost, we have to find the most efficient combination of those two test sets without sacrificing the test quality. We describe methods for finding the optimal combination of pseudorandom and deterministic test sets of the whole system, consisting of multiple cores, under given memory constraints, so that the total test time is minimized. Our approach employs a fast estimation methodology in order to avoid exhaustive search and to speed up the calculation process. Experimental results have shown the efficiency of the algorithms to find a near-optimal solutions.

**Key words:** digital test, hybrid BIST, system-on-chip.

## 1. INTRODUCTION

Rapid advances of the microelectronics technology in recent years have brought new possibilities to the design and manufacturing of integrated circuits (ICs) [[1]]. Nowadays many systems are designed by embedding pre-designed and pre-verified complex functional blocks, usually referred as cores, into one single die. Such core-based design technique has led to increased design productivity, but at the same time it has introduced additional test-related problems. These additional testing problems, together with the test problems induced due to the

complexity and heterogeneous nature of such systems-on-chip (SoC), pose great challenges to the SoC testing community [2]. Typically, a SoC consists of micro-processor cores, digital logic blocks, analogue devices, and memory structures [3]. These different types of components were traditionally tested as separate chips by dedicated automatic test equipment of different types. Now they must be tested all together as a single chip either by a super tester, which is capable of handling different types of cores and is very expensive, or by multiple testers, which is very time-consuming due to the time of moving from one tester to another.

Another key issue to be addressed for SoC testing is the implementation of test access mechanisms on the chip. For traditional system-on-board design, direct test access to the peripheries of the basic components, in the form of separate chips, is usually available. For the corresponding cores, embedded deeply in a SoC, such access is impossible. Therefore, additional test access mechanisms must be included in a SoC to connect the core peripheries to the test sources and sinks, which are the SoC pins when testing by an external tester.

Many testing problems, discussed above, can be overcome by using a built-in self-test (BIST) strategy. For example, the test access cost can be substantially reduced by putting the test sources and sinks next to the cores to be tested. BIST can also be used to deal with the discrepancy between the speed of the SoC, which is increasing rapidly, and that of the tester, which will soon be too slow to match typical SoC clock frequencies. The introduction of BIST mechanisms in a SoC will also improve the diagnostic ability and field-test capability, which are essential for many applications where regular operation and maintenance test is needed [4].

Since the introduction of BIST mechanisms into a SoC is a complex task, we need to develop powerful automated design methods and tools to optimize the test function together with other design criteria as well as to speed up the design process. In this paper, we are going to concentrate on one of the improvements of the classical BIST approach, namely on the hybrid BIST. We will describe the basic concepts of the approach and propose optimization methods for satisfying different test constraints.

## 2. RELATED WORK

A classical BIST architecture consists of a test pattern generator (TPG), a test response analyser (TRA) and a BIST control unit (BCU), all implemented on the chip. Different implementations of such BIST architectures have been available and some of them have wide acceptance. One of the major problems of the classical BIST implementations is related to the TPG design. Typically, such a TPG is implemented by linear feedback shift registers (LFSR) [5–7]. Since the test patterns, generated by an LFSR, are pseudorandom by nature and have linear dependencies [8], the LFSR-based approach often does not guarantee sufficiently

high fault coverage (especially in the case of large and complex designs), and demands very long test application times in addition to high overheads. Therefore, several proposals have been made to combine pseudorandom test patterns, generated by LFSRs, with deterministic patterns [9–15], to form a mixed-mode solution.

A mixed-mode scheme uses pseudorandom patterns to cover easy-to-detect faults and, subsequently, deterministic patterns to target the remaining hard-to-detect faults. The main strength of these approaches is in the possibility to have a trade-off between test data storage and test application time by varying the ratio of pseudorandom and deterministic test patterns.

One of the mixed-mode approaches is based on the LFSR reseeding. In this approach, the quality of the test sequence is improved by generating only a limited number of test patterns from one LFSR seed (initial state), and during the test generation process the LFSR is reseeded with new seeds. This idea was first proposed by Koenemann in 1991 [12]. These new seeds are used to generate pseudorandom sequences and to encode the deterministic test patterns in order to reduce the number of non-useful patterns. In this approach, only a set of LFSR seeds have to be stored instead of the complete set of patterns, and as a result less storage is needed.

Several heuristic approaches have been proposed to identify multiple seeds, and the number of vectors applied starting with each seed, to minimize the overall test application time under a given constraint on the maximum number of seeds [10,15]. If a small LFSR is used, it may not always be possible to find a seed that will generate a required deterministic test pattern, hence the fault coverage may remain low. Therefore, a different reseeding scenario, based on multiple-polynomial LFSRs, has been proposed in [16]. There, deterministic patterns are encoded with a number of bits, specifying a seed and a polynomial identifier. During testing, not only the appropriate seed, but also the corresponding feedback polynomial, have to be loaded into the LFSR. Another alternative is to use variable-length seeds [15]. However, all these techniques generate test sets of excessive length.

Another class of mixed-mode schemes embeds deterministic test patterns into LFSR sequences by mapping LFSR states to deterministic test patterns. This can be achieved by adding extra circuitry to generate control signals that complement certain bits or fix them either as 0 or 1 [17]. A hardware for implementing the bit-flipping or bit-fixing sequence generation logic is the major cost of this approach, as it has to be customized for a given CUT and LFSR. An alternative approach transforms the LFSR-generated patterns into a new set of test patterns with higher fault coverage. The transformation is carried out by a mapping logic, which decodes sets of ineffective patterns and maps them into vectors that detect the hard-to-test faults [9]. The outputs of an $n$-stage random TPG are input to a mapping logic and the outputs of the mapping logic drive the inputs of the CUT. Nevertheless, most of these variations of controlling the bits of the LFSR sequence have not yet solved the problems with random resistance.

The main objective of all these previously mentioned methods has been improvement of the test quality in terms of fault coverage, while different aspects related to the test cost, like test length, area overhead and tester memory requirements, were largely omitted or handled in isolation. In this paper an alternative approach, called hybrid BIST, will be described. In particular, different test optimization algorithms, based on the proposed hybrid BIST architecture, will be presented.

## 3. HYBRID BIST

As described earlier, a typical self-test approach employs usually some form of pseudorandom test pattern generators. These test sequences are often very long and not sufficient to detect all the faults. To avoid the test quality loss due to random pattern resistant faults and to speed up the testing process, we can apply deterministic test patterns targeting the random resistant and difficult to test faults. Such a hybrid BIST approach starts usually with a pseudorandom test sequence of length $L$. After the application of pseudorandom patterns, a stored test approach will be used [18]. For the stored test approach, pre-computed test patterns are applied to the core under test in order to reach the desirable fault coverage level. For off-line generation of deterministic test patterns, arbitrary software test generators may be used based on, for example, deterministic or genetic algorithms.

In a hybrid BIST technique, the length of the pseudorandom test is an important design parameter, which determines the behaviour of the whole test process. A shorter pseudorandom test sequence implies a larger deterministic test set. This requires additional memory space, but at the same time, shortens the overall test time. A longer pseudorandom test, on the other hand, will lead to larger test application time with reduced memory requirement. Therefore it is crucial to determine the optimal length of pseudorandom test in order to minimize the total testing cost.

Figure 1 illustrates graphically the total cost of a hybrid BIST, consisting of pseudorandom test patterns and stored test patterns generated off-line. The horizontal axis in Fig. 1 denotes the fault coverage, achieved by the pseudorandom test sequence before switching from the pseudorandom test to the stored one. Zero pseudorandom test coverage is the case when only stored test patterns are used and therefore the cost of stored test is the biggest at this point. The figure illustrates the situation where 100% fault coverage is achievable with pseudorandom vectors alone.

The total test cost of the hybrid BIST $C_{\text{TOTAL}}$ can therefore be defined as

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

where $C_{\text{GEN}}$ is the cost related to the effort for generating $L$ pseudorandom test patterns (number of clock cycles), $C_{\text{MEM}}$ is related to the memory cost for storing

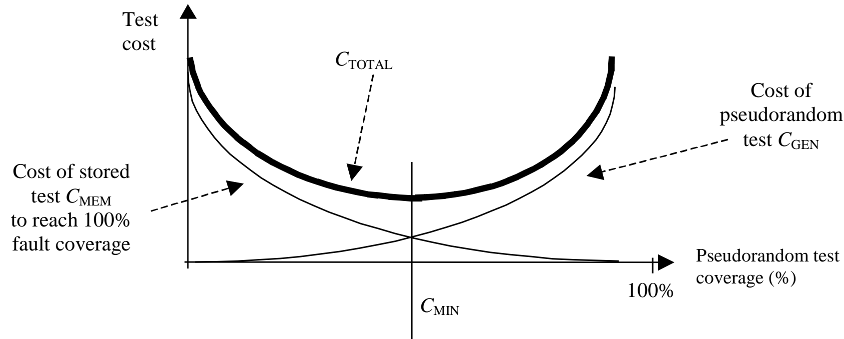



**Fig. 1.** Cost calculation for the hybrid BIST (under 100% assumption).

$S$ pre-computed test patterns to improve the pseudorandom test set, and $\alpha$ and $\beta$ are constants to map the test length and memory space to the costs of the two parts of the test solutions.

We should note that defining the test cost as a sum of two costs, the cost of time for the pseudorandom test generation and the cost of memory associated with storing the TPG produced test, is a rather simplified cost model for the hybrid BIST technique. In this simplified model, neither the basic cost of silicon (or its equivalent), occupied by the LFSR-based generator, nor the effort, needed for generating deterministic test patterns, are taken into account. Similarly, all aspects related to test data transportation are omitted. However, these aspects can easily be added to the cost calculation formula after the desired hardware architecture and deterministic test pattern generation approaches are chosen. In the following sections, we are going to provide the algorithms to find the best trade-off between the length of pseudorandom test sequence and the number of deterministic patterns. For making such a trade-off, the basic implementation costs are invariant and will not influence the optimal selection of the hybrid BIST parameters.

On the other hand, the attempt to add "time" to "space" (even in terms of their cost) seems rather controversial as it is very hard to specify which one costs more in general (or even in particular cases) and how to estimate these costs. This is also the reason why the total cost of the BIST function is not considered here. The values of the parameters $\alpha$ and $\beta$ in the cost function are left to be determined by the designer and can be seen as one of the design decisions. If needed, it is possible to separate these two costs (time and memory space) and consider, for example, one of them as a design constraint.

Figure 1 illustrates also how the cost of pseudorandom test is increasing when striving to higher fault coverage (the $C_{GEN}$ curve). In general, it can be very expensive to achieve high fault coverage with pseudorandom test patterns alone. The $C_{MEM}$ curve describes the cost that we have to pay for storing additional pre-computed tests at the given fault coverage level, reached by pseudorandom testing. The total cost $C_{TOTAL}$ is the sum of the above two costs. The $C_{TOTAL}$ curve is shown in Fig. 1, where the minimum point is marked as $C_{MIN}$.

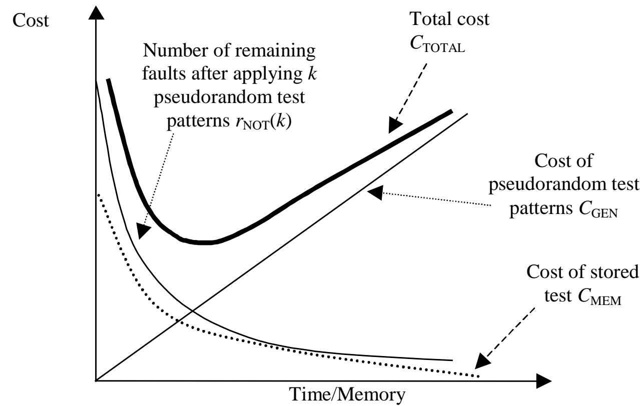



**Fig. 2.** Cost calculation for the hybrid BIST.

As mentioned earlier, in many situations 100% fault coverage is not achievable with only pseudorandom vectors. Therefore we have to include this assumption to the total cost calculation. The situation is illustrated in Fig. 2, where the horizontal axis indicates the number of pseudorandom patterns applied, instead of the fault coverage level. The curve of the total cost $C_{\text{TOTAL}}$ is still the sum of two cost curves $C_{\text{GEN}} + C_{\text{MEM}}$ with the new assumption that the maximum fault coverage is achievable only by either the hybrid BIST or pure deterministic test.

## 4. HYBRID BIST ARCHITECTURES

The previous section described the basic principles of the hybrid BIST and introduced the test cost calculation formulas. In this section, some basic concepts of hybrid BIST architectures will be discussed. Although our optimization methods are not devised for a particular test architecture and different architectural assumptions can easily be incorporated into the algorithms, some basic assumptions have to be made.

### 4.1. Core-level hybrid BIST architecture

We have divided cores into two large classes. To the first class belong the cores that are equipped with their own pseudorandom test pattern generator and only deterministic patterns have to be transported to the cores. The second class consists of cores with no pre-existing BIST structures. Such cores require an alternative approach, where pseudorandom and deterministic test patterns have to be transported to the core under test from external sources.

At the core level, pseudorandom testing can be performed using many different scenarios, as described earlier. We have assumed a core-level hybrid BIST architecture that is depicted in Fig. 3, where the pseudorandom pattern

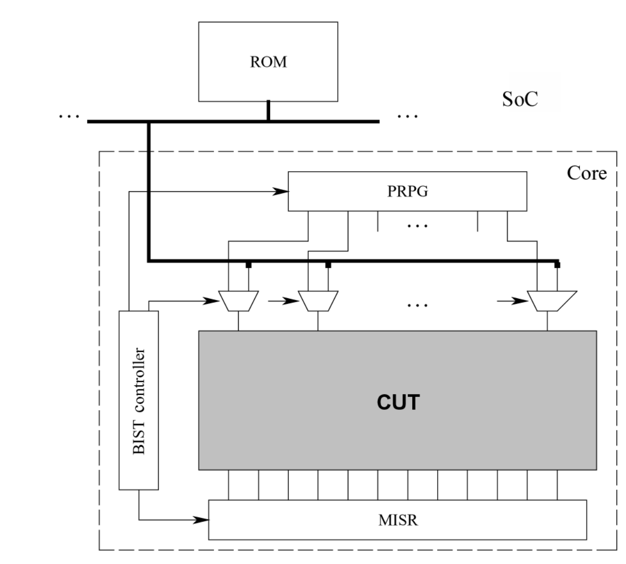



**Fig. 3.** Hardware-based core-level hybrid BIST architecture.

generator (PRPG) and the Multiple Input Signature Analyser (MISR) are implemented inside the core under test (CUT) using LFSRs or any other structure that provides pseudorandom test vectors with a required degree of randomness. The deterministic test patterns are precomputed off-line and stored outside the core, either in a ROM or in an ATE [19].

Core test is performed in two consecutive stages. During the first stage, pseudorandom test patterns are generated and applied. After a predetermined number of test cycles, additional test is performed with deterministic test patterns from the memory. For combinatorial cores, where a test-per-clock scheme can be used, each primary input of the CUT has a multiplexer at the input that determines whether the test is coming from the PRPG or from the memory (Fig. 3). The response is compacted into the MISR in both cases. The architecture can easily be modified with no or only minor modification of the optimization algorithms to be presented in the following sections.

As testing of sequential cores is very complex, it is assumed here that every sequential core contains one or several scan paths (full scan). Therefore a test-per-scan scheme has to be used and, for every individual core, the "Self-Test Using MISR and Parallel Shift Register Sequence Generator" (STUMPS) [20] architecture is assumed. Both internally generated pseudorandom patterns and externally stored deterministic test patterns are therefore applied via scan chains. In both situations, every core's BIST logic is capable of producing a set of independent pseudorandom test patterns, i.e. the pseudorandom test sets for all the cores can be carried out simultaneously and independently.

## 4.2. System-level hybrid BIST architectures

*4.2.1. Parallel hardware-based hybrid BIST architecture*

We start with a system-level test architecture, where every core has its own dedicated BIST logic. The deterministic tests are applied from the external source (either on-chip memory or ATE), one core at a time; in the current approach we have assumed for test data transportation an AMBA-like test bus [[21]]. AMBA (Advanced Microcontroller Bus Architecture) integrates an on-chip test access technique that reuses the basic bus infrastructure. An example of a multi-core system with such a test architecture is given in Fig. 4.

Our optimization methods are not dependent of the location of the deterministic test patterns. These patterns can be applied either from the external ATE or from an on-chip memory (ROM). As we have assumed a bus-based test architecture, the time needed for test data transportation from the particular test source to a given CUT is always the same. The corresponding time overhead, related to the test data transportation, can easily be incorporated into the proposed algorithms.

Considering the assumed test architecture, only one deterministic test set can be applied at any given time, while any number of pseudorandom test sessions can take place in parallel. To enforce the assumption that only one deterministic test can be applied at a time, a simple ad-hoc scheduling can be used.

The above type of architecture, however, may not always be feasible as not all cores may be equipped with self-test structures. It may also introduce a significant area overhead and performance degradation, as some cores may require excessively large LFSRs.

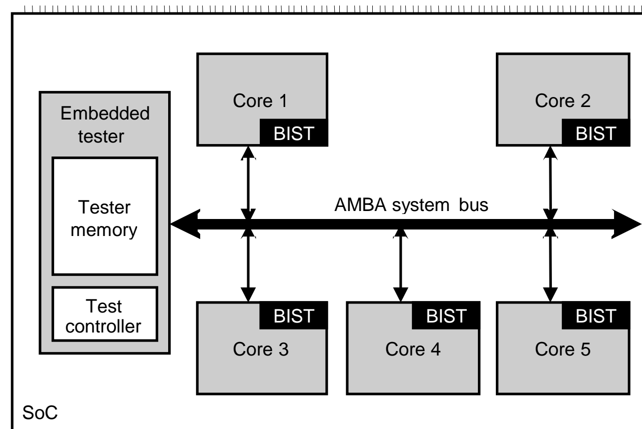



**Fig. 4.** An example of a core-based system with independent BIST resources.

#### *4.2.2. Software-based hybrid BIST architecture*

To make the BIST approach more attractive, we have to tackle the hardware overhead problem and to find solutions to reduce the additional delay and the long test application times. At the same time, fault coverage has to be kept at a high level. The simplest and most straightforward solution is to replace the hardware LFSR implementation 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). The software-based approach, on the other hand, is criticized because of the large memory requirements, as we have to store the test program and some test patterns, which are required for initialization and reconfiguration of the self-test cycle [11]. However, some preliminary results regarding such an approach for PCBs have been reported in [22] and show that a software-based approach is feasible.

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

In the test mode, the test program will be executed by 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. This approach, however, may lead to a more complex test controller, as every core requires pseudorandom patterns with different characteristics (polynomial, initial state and length, for example). The general concept of the software based pseudorandom TPG is depicted in Fig. 5.

As the LFSR is implemented in software, there are no hardware constraints for the actual implementation. This allows developing for each particular core an efficient pseudorandom scheme without concerning about the hardware cost except the cost for the ROM. As has been shown by experiments, the selection of the best possible pseudorandom scheme is an important factor for such an approach [11].

As discussed in [11], the program to emulate the LFSR can be very simple and therefore the memory requirements for storing the pseudorandom TPG program together with the LFSR parameters are relatively small. This, however, does not have any influence on the cost calculation and optimization algorithms, to be proposed. These algorithms are general and can be applied to the hardware-based as well as to the software-based hybrid BIST optimization.

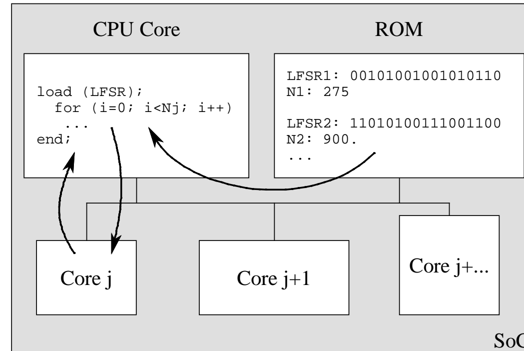



**Fig. 5.** LFSR emulation.

## 5. COST CALCULATION FOR HYBRID BIST

For hybrid BIST cost calculations we have to calculate costs of the pseudorandom test $C_{\mathrm{GEN}} = \alpha L$ and deterministic test $C_{\mathrm{MEM}} = \beta S$. Creating the curve $C_{\mathrm{GEN}}$ is not difficult. For this purpose, a simulation of the behaviour of the LSFR, used for pseudorandom test pattern generation, is needed. Fault simulation should be carried out for the complete test sequence, generated by the LFSR. As a result of such a simulation, we find for each clock cycle the list of faults, which were covered up to this clock cycle. By removing these faults from the complete fault list, we know the number of faults remaining to be tested.

More difficult is to find the values of $\beta S$, the cost for storing additional deterministic patterns in order to reach the given fault coverage level (100% in the ideal case). In [18] we proposed a method based on repetitive use of the ATPG and in [23] a method based on fault table manipulations was described. Both procedures are accurate but time-consuming and therefore not feasible for larger designs.

To overcome the complexity explosion problem we have developed an estimation methodology [24] that leads us to the approximate solution. This can be used as an initial solution for the search of more accurate results, using different optimization heuristics, like Simulated Annealing [25]. In [26], a method based on Tabu search [27] has been proposed.

Let us denote the deterministic test set by $TD$ and efficient pseudorandom test set [28] by $TPE$. In the following we will use $FD(i)$ and $FPE(i)$ to denote the fault coverage figures of the test sequences $TD(i)$ and $TPE(i)$, respectively, where $i$ is the length of the test sequence.

**Procedure 1. Estimation of the length of the deterministic test set TD**

1. Calculate, by fault simulation, the fault coverage functions $FD(i)$, $i = 1, 2, \ldots, |TD|$, and $FPE(i)$, $i = 1, 2, \ldots, |TPE|$. The patterns in $TD$ are ordered in such a way that each pattern, put into the sequence, contributes with maximum increase in fault coverage.

2. For each $i^* \le |TPE|$, find the fault coverage value $F^*$ that can be reached by a sequence of patterns $(P_1, P_2, \ldots, P_{i^*}) \subseteq TPE$ (Fig. 6).
3. By solving the equation $FD(i) = F^*$, find the maximum integer value $j^*$ that satisfies the condition $FD(j^*) \le F^*$. The value of $j^*$ is the length of the deterministic sequence that can achieve the same fault coverage $F^*$.
4. Calculate the value of $|TD^E(i^*)| = |TD| - j^*$, which is the number of test patterns needed for the $TD$ to reach the maximum achievable fault coverage.

The value $|TD^E(i^*)| = |TD| - j^*$, calculated by the Procedure 1, can be used to estimate the length of the deterministic test sequence $TD^*$ in the hybrid test set $TH = \{TP^*, TD^*\}$ with $i^*$ efficient test patterns in $TP^*$. By finding $|TD^E(j)|$ for all $j = 1, 2, \ldots, |TPE|$ we get the cost function estimate $C^E_{\text{MEM}}(j)$.

In the following we shall illustrate the Procedure 1 with an example. In Fig. 7 we have presented an extract of fault simulation results for both test sets ($FC$ is fault coverage). The length of the pseudorandom sequence has to be only so long as potentially necessary. By knowing the length of the complete deterministic test set and fault coverage figures for every individual pattern, we can estimate the size of the additional deterministic test set for any length of the pseudorandom test sequence, as illustrated in Fig. 7. We can see that for a given core, 60 deterministic test cycles are needed to obtain the same fault coverage as with 524

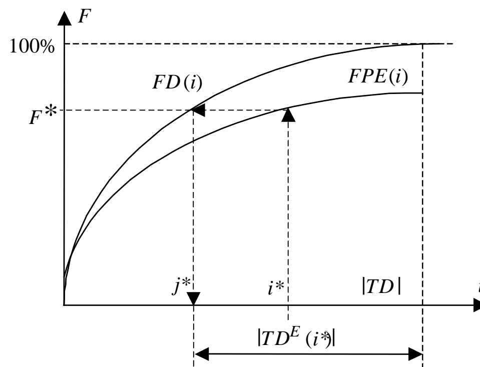
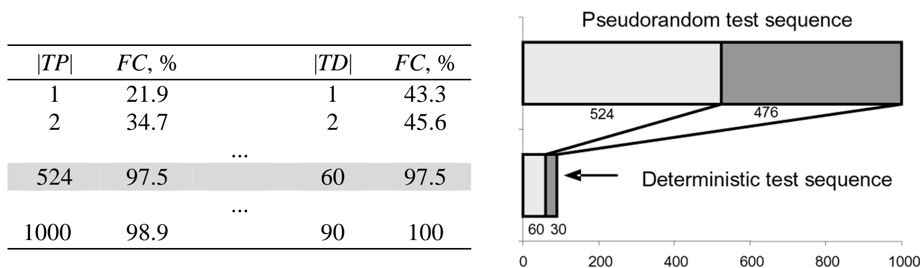



**Fig. 6.** Estimation of the length of the deterministic test sequence.



| $\|TP\|$ | $FC$, % | | $\|TD\|$ | $FC$, % |
|---|---|---|---|---|
| 1 | 21.9 | | 1 | 43.3 |
| 2 | 34.7 | | 2 | 45.6 |
| | | ... | | |
| 524 | 97.5 | | 60 | 97.5 |
| | | ... | | |
| 1000 | 98.9 | | 90 | 100 |



**Fig. 7.** Estimation of the length of the deterministic test sequence.

pseudorandom test cycles and it requires additional 30 deterministic test cycles to reach 100% fault coverage. Based on this information we assume that if we apply those 30 deterministic test cycles on top of the 524 pseudorandom cycles, we can obtain close to the maximum fault coverage.

We have demonstrated [24] that this estimation methodology can efficiently be used in different test cost minimization algorithms. In the following we shall use it for test time minimization in the multi-core environment.

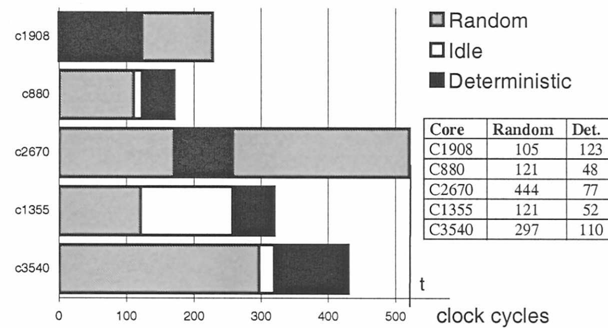
## 6. HYBRID BIST IN THE SoC ENVIRONMENT

Many publications are devoted to the testing of core-based systems [13,29–34]. So far the main emphasis has been on the test scheduling, TAM design and testability analysis. The earlier test scheduling work has had the objective to determine start times for each test so that the total test application time is minimized. This assumes a fixed set of tests and test resources together with a test access architecture. Some approaches take into account also test conflicts and different constraints, e.g. power. However, there have not been investigations to find the optimal test sets for testing every individual core in such a manner that the test time of the total system is minimized and different ATE constraints satisfied.

As total cost minimization for multi-core systems is an extremely complex problem and is rarely used in practice then the main emphasis here is on test time minimization under memory constraints. The memory constraints can be seen as limitations of the on-chip memory or automatic test equipment, where the deterministic test set will be stored, and are therefore of great practical importance. We shall concentrate on the test architecture, where every core is equipped with its own pseudorandom pattern generator and only deterministic patterns have to be transported to the cores (Fig. 4).

It is important to mention here that the following approach neither takes into account the test power nor do we propose any methods for test access mechanism optimization. Those problems can be solved after the efficient test set for every individual core has been developed [35] and therefore are not considered here.

In order to explain the test time minimization problem for multi-core systems, let us use an example design, consisting of 5 cores, each core as a different ISCAS benchmark. Using the hybrid BIST optimization methodology [23] we can find the optimal combination between pseudorandom and deterministic test patterns for every individual core (Fig. 8). Considering the assumed test architecture, only one deterministic test set can be applied at any given time, while any number of pseudorandom test sessions can take place in parallel. To enforce the assumption that only one deterministic test can be applied at a time, a simple *ad hoc* scheduling method can be used. The result of this schedule defines the starting moments for every deterministic test session, the memory requirements, and the total test length $t$ for the whole system. This situation is illustrated in Fig. 8.




| Core | Random | Det. |
|---|---|---|
| C1908 | 105 | 123 |
| C880 | 121 | 48 |
| C2670 | 444 | 77 |
| C1355 | 121 | 52 |
| C3540 | 297 | 110 |

**Fig. 8.** *Ad hoc* test schedule for a hybrid BIST of the core-based system example.

As it can be seen in Fig. 8, the solution, where every individual core has the best possible combination between pseudorandom and deterministic patterns, usually does not lead to the best system-level test solution. In this example, we have illustrated three potential problems:

- the total test length of the system is determined by the single longest individual test set, while other tests may be substantially shorter;
- the resulting deterministic test sets do not take into account the memory requirements, imposed by the size of the on-chip memory or the external test equipment;
- the proposed test schedule may introduce idle periods, due to the scheduling conflicts between the deterministic tests of different cores.

There are several possibilities for improvement. For example, the *ad hoc* solution in Fig. 8 can easily be improved by using a better scheduling strategy. This, however, does not necessarily lead to a significantly better solution as the ratio between pseudorandom and deterministic test patterns for every individual core is not changed. Therefore we have to explore different combinations between pseudorandom and deterministic test patterns for every individual core in order to find a solution, where the total test length of the system is minimized and the memory constraints are satisfied. In the following sections, we shall define this problem more precisely and describe a fast iterative algorithm for calculating the optimal combination between different test sets for the whole system.

### 6.1. Basic definitions and formulation of the problem

Let us assume that a system $S$ consists of $n$ cores $C_1, C_2, \ldots, C_n$. For every core $C_k \in S$ a complete sequence of deterministic test patterns $TD_k^F$ and a complete sequence of pseudorandom test patterns $TP_k^F$ can be generated.

**Definition 1.** A *hybrid BIST* set $TH_k = \{TP_k, TD_k\}$ for a core $C_k$ is a sequence of tests, constructed from a subset $TP_k \subseteq TP_k^F$ of the pseudorandom test sequence and a deterministic test sequence $TD_k \subseteq TD_k^F$. The sequences $TP_k$ and $TD_k$ complement each other to achieve the maximum achievable fault coverage.

**Definition 2.** A pattern in a pseudorandom test sequence is called *efficient* if it detects at least one new fault that is not detected by the previous test patterns in the sequence. The ordered sequence of efficient patterns form an *efficient pseudorandom test sequence* $TPE_k = (P_1, P_2, \ldots, P_n) \subseteq TP_k$. Each efficient pattern $P_j \in TPE_k$ is characterized by the length of the pseudorandom test sequence $TP_k$, from the start to the efficient pattern $P_j$, including $P_j$. An efficient pseudorandom test sequence $TPE_k$, which includes all efficient patterns of $TP_k^F$ is called *full efficient pseudorandom test sequence* and denoted by $TPE_k^F$.

**Definition 3.** The cost of a hybrid test set $TH_k$ for a core $C_k$ is determined by the total length of its pseudorandom and deterministic test sequences, which can be characterized by their costs, $COST_{P,k}$ and $COST_{D,k}$, respectively:

$$COST_{T,k} = COST_{P,k} + COST_{D,k} = \sigma |TP_k| + \varphi_k |TD_k|, \quad (2)$$

and by the cost of recourses needed for storing the deterministic test sequence $TD_k$ in the memory:

$$COST_{M,k} = \gamma_k |TD_k|. \quad (3)$$

The parameters $\sigma$ and $\varphi_k (k = 1, 2, \ldots, n)$ can be introduced by the designer to align the application times of different test sequences. For example, when a test-per-clock BIST scheme is used, a new test pattern can be generated and applied in each clock cycle and in this case $\sigma = 1$. The parameter $\varphi_k$ for a particular core $C_k$ is equal to the total number of clock cycles needed for applying one deterministic test pattern from the memory. In a special case, when deterministic test patterns are applied by an external test equipment, application of deterministic test patterns may be up to one order of magnitude slower than by applying BIST patterns. The coefficient $\gamma_k$ is used to map the number of test patterns in the deterministic test sequence $TD_k$ into the memory recourses, measured in bits.

**Definition 4.** When assuming the test architecture described above, a hybrid test set $TH = \{TH_1, TH_2, \ldots, TH_n\}$ for a system $S = \{C_1, C_2, \ldots, C_n\}$ consists of hybrid tests $TH_k$ for each individual core $C_k$, where the pseudorandom components of $TH$ can be scheduled in parallel, whereas the deterministic components of $TH$ must be scheduled in sequence due to the shared test resources.

**Definition 5.** $J = (j_1, j_2, \ldots, j_n)$ is called the *characteristic vector* of a hybrid test set $TH = \{TH_1, TH_2, \ldots, TH_n\}$, where $j_k = |TPE_k|$ is the length of the efficient pseudorandom test sequence $TPE_k \subseteq TP_k \subseteq TH_k$. According to Definition 2, for each $j_k$ corresponds a pseudorandom subsequence $TP_k(j_k) \subseteq TP_k^F$, and according to Definition 1, any pseudorandom test sequence $TP_k(j_k)$ should be complemented with a deterministic test sequence, denoted with $TD_k(j_k)$, that is generated in order to achieve the maximum achievable fault coverage. Based

on this we can conclude that the characteristic vector $J$ determines entirely the structure of the hybrid test set $TH_k$ for all cores $C_k \in S$.

**Definition 6.** The test length of a hybrid test $TH = \{TH_1, TH_2, \ldots, TH_n\}$ for a system $S = \{C_1, C_2, \ldots, C_n\}$ is given by:

$$COST_T = \max\{\max_k(\sigma |TP_k| + \varphi_k |TD_k|), \sum_k \varphi_k |TD_k|\}. \quad (4)$$

The total cost of resources, needed for storing the patterns from all deterministic test sequences $TD_k$ in the memory, is given by

$$COST_M = \sum_k COST_{M,k}. \quad (5)$$

**Definition 7.** Let us introduce a generic cost function $COST_{M,k} = f_k(COST_{T,k})$ for every core $C_k \in S$, and an integrated generic cost function $COST_M = f_k(COST_T)$ for the whole system $S$. The functions $COST_{M,k} = f_k(COST_{T,k})$ will be created in the following way. Let us have a hybrid BIST set $TH_k(j) = \{TP_k(j), TD_k(j)\}$ for a core $C_k$ with $j$ efficient patterns in the pseudorandom test sequence. By calculating the costs $COST_{T,k}$ and $COST_{M,k}$ for all possible hybrid test set structures $TH_k(j)$, i.e. for all values $j = 1, 2, \ldots, |TPE_k^F|$, we can create the cost functions $COST_{T,k} = f_{T,k}(j)$ and $COST_{M,k} = f_{M,k}(j)$. By taking the inverse function $j = f_{T,k}^{-1}(COST_{T,k})$, and inserting it into the $f_{M,k}(j)$ we get the generic cost function $COST_{M,k} = f_{M,k}(f_{T,k}^{-1}(COST_{T,k})) = f_k(COST_{T,k})$ where the memory costs are directly related to the lengths of all possible hybrid test solutions. The integrated generic cost function $COST_M = f(COST_T)$ for the whole system is the sum of all cost functions $COST_{M,k} = f_k(COST_{T,k})$ of individual cores $C_k \in S$.

From the function $COST_M = f(COST_T)$ the value of $COST_T$ for every given value of $COST_M$ can be found. The value of $COST_T$ determines the lower bound of the length of the hybrid test set for the whole system. To find the component $j_k$ of the characteristic vector $J$, i.e. to find the structure of the hybrid test set for all cores, the equation $f_{T,k}(j) = COST_T$ should be solved.

The objective here is to find a shortest possible $(\min(COST_T))$ hybrid test sequence $TH_{\mathrm{OPT}}$ when the memory constraints are not violated i.e., $COST_M \leq COST_{M,\mathrm{LIMIT}}$.

### 6.2. Minimization of the test length under memory constraints

As described above, the exact calculations for finding the cost of the deterministic test set $COST_{M,k} = f_k(COST_{T,k})$ are very time-consuming. Therefore, we shall use the cost estimates, calculated by Procedure 1, instead. Using estimates can give us a close to minimal solution for the test length of the hybrid test at given memory constraints. After obtaining this solution, the cost estimates

can be improved and another, better solution can be calculated. This iterative procedure will be continued until we reach the final solution.

**Procedure 2. Test length minimization**

1. Given the memory constraint $COST_{M,\text{LIMIT}}$, find the estimated total test length $COST_T^{E*}$ as a solution to the equation $f^E(COST_T^E) = COST_{M,\text{LIMIT}}$.
2. Based on $COST_T^{E*}$, find a candidate solution $J^* = (j_1^*, j_2^*, \ldots, j_n^*)$ where each $j_k^*$ is the maximum integer value that satisfies the equation $COST_{T,k}^E(j_k^*) \leq COST_T^{E*}$.
3. To calculate the exact value of $COST_M^*$ for the candidate solution $J^*$, find the set of not yet detected faults $F_{\text{NOT},k}(j_k^*)$ and generate the corresponding deterministic test set $TD_k^*$ by using an ATPG algorithm.
4. If $COST_M^* = COST_{M,\text{LIMIT}}$, go to the Step 9.
5. If the difference $|COST_M^* - COST_{M,\text{LIMIT}}|$ is bigger than that in the earlier iteration, make a correction $\Delta t = \Delta t/2$ and go to Step 7.
6. Calculate a new test length $COST_T^{E,N}$ from the equation $f_k^E(COST_T^E) = COST_M^*$, and find the difference $\Delta t = COST_T^{E,*} - COST_T^{E,N}$.
7. Calculate a new cost estimate $COST_T^{E,*} = COST_T^{E,*} + \Delta t$ for the next iteration.
8. If the value of $COST_T^{E,*}$ is the same as in an earlier iteration, go to Step 9, otherwise go to Step 2.
9. *END*: The vector $J^* = (j_1^*, j_2^*, \ldots, j_n^*)$ is the solution.

To illustrate the above procedure, in Figs. 9 and 10 an example of the iterative search for the shortest length of the hybrid test is given. Figure 9 represents all the basic cost curves $COST_{D,k}^E(j)$, $COST_{P,k}^E(j)$, and $COST_{T,k}^E(j)$, as functions of the length $j$ of $TPE_k$ where $j_{\min}$ denotes the optimal solution for a single core hybrid BIST optimization problem [18]. Figure 10 represents the estimated

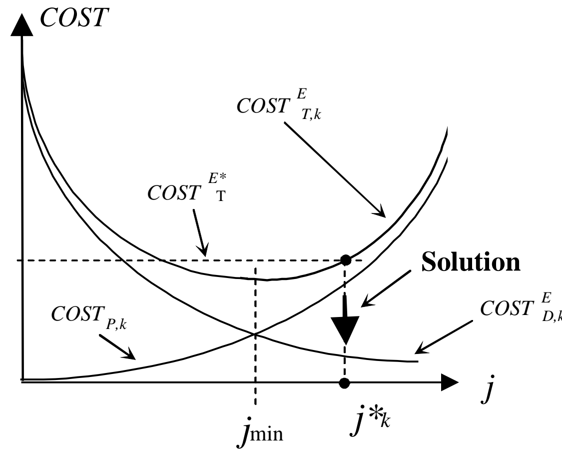



**Fig. 9.** Cost curves for a given core $C_k$.

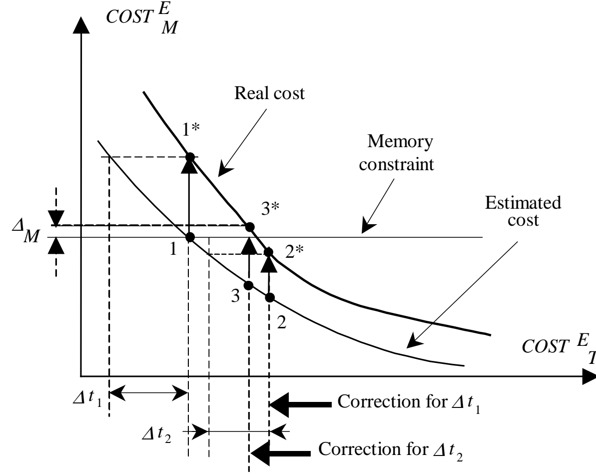



**Fig. 10.** Minimization of the test length.

generic cost function $COST_M^E = f^E(COST_T^E)$ for the whole system. At first (Step 1), the estimated $COST_T^{E*}$ for the given memory constraints is found (point 1 in Fig. 10). Then (Step 2), based on $COST_T^{E*}$ the length $j_k^*$ of $TPE_k$ for the core $C_k$ in Fig. 9 is found. This procedure (Step 2) is repeated for all the cores to find the characteristic vector $J^*$ of the system as the first iterative solution. After that the real memory cost $COST_M^{E*}$ is calculated (Step 3, point 1* in Fig. 10). As we see in Fig. 10, the value of $COST_M^{E*}$ at the point 1* violates the memory constraints. The difference $\Delta t_1$ is determined by the curve of the estimated cost (Step 6). After correction, a new value of $COST_T^{E*}$ is found (point 2 in Fig. 10). Based on $COST_T^{E*}$, a new $J^*$ is found (Step 2), and a new $COST_M^{E*}$ is calculated (Step 3, point 2* in Fig. 10). An additional iteration via points 3 and 3* can be followed in Fig. 10.

It is easy to see that Procedure 2 always converges. By each iteration we get closer to the memory constraints level, and also closer to the minimal test length at given constraints. However, the solution may be only near-optimal since we only evaluate solutions, derived from the estimated cost functions.

## 7. EXPERIMENTAL RESULTS

We have performed experiments with several systems, composed of different ISCAS benchmarks [36] as cores (S1: c5315, c880, c432, c499, c499, c5315; S2: c432, c499, c880, c1355, c1908, c5315, c6288; S3: c880, c5315, c3540, c1908, c880), using our in-house software tools [37,38]. The results are presented in Table 1. In Table 1 our approach, where the test length is found based on estimates, is compared with an approach, where deterministic test sets have been found by manipulating the fault tables for every possible switching point between

pseudorandom and deterministic test patterns. As it can be seen from the results, our approach can give significant speedup (more than one order of magnitude), while retaining acceptable accuracy (the biggest deviation is less than 9% from the fault table based solution, and in average 2.4%).

In Fig. 11 the estimated cost curves for the individual cores and the estimated and real cost curves for one of the systems with 7 cores are shown. In Fig. 11 is also shown a test solution point for this system under given memory constraint that has been found based on our algorithm. In this example a memory constraint

**Table 1.** Experimental results with combinatorial cores

| System | Number of cores | Memory constraint, bits | Fault table based approach | | Our approach | |
|---|---|---|---|---|---|---|
| | | | Total test length, clocks | CPU time*, s | Total test length, clocks | CPU time, s |
| S1 | 6 | 20 000 | 222 | 3 772.84 | 223 | 199.78 |
| | | 10 000 | 487 | | 487 | 57.08 |
| | | 7 000 | 552 | | 599 | 114.16 |
| S2 | 7 | 14 000 | 207 | 3 433.10 | 209 | 167.3 |
| | | 5 500 | 540 | | 542 | 133.84 |
| | | 2 500 | 1017 | | 1040 | 200.76 |
| S3 | 5 | 7 000 | 552 | 10 143.14 | 586 | 174.84 |
| | | 3 500 | 3309 | | 3413 | 291.40 |
| | | 2 000 | 8549 | | 487 | 199.78 |

* CPU time for calculating all possible hybrid BIST solutions.

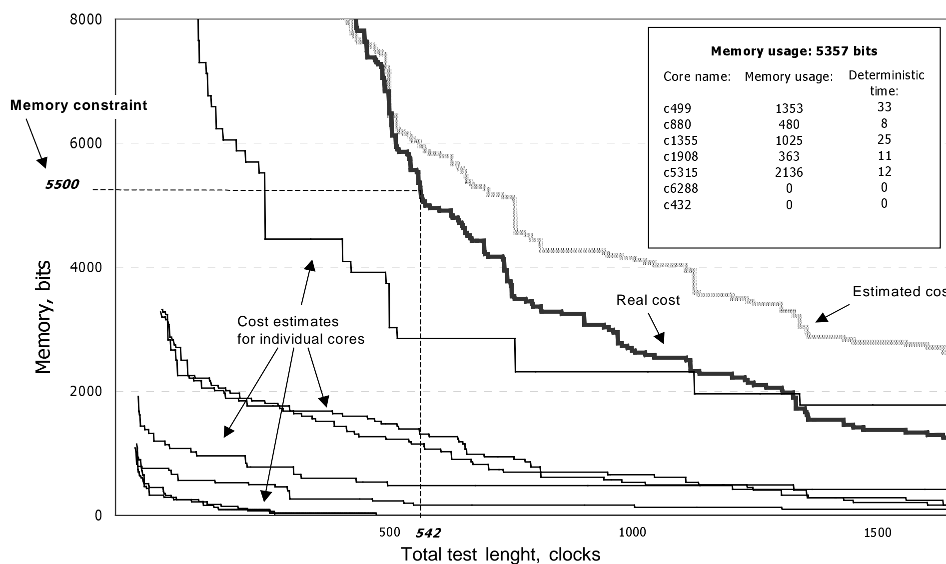



**Fig. 11.** The final test solution for the system S2 ($M_{\mathrm{LIMIT}}$ = 5 500).

$M_{LIMIT}$ = 5500 bits has been used. The final test length for this memory constraint is 542 clock cycles and that results in a test schedule depicted in Fig. 12. In Fig. 13 another test schedule for the same system, when the memory constraints are different ($M_{LIMIT}$ = 14 000 bits), is shown.

This approach can easily be extended to systems with full-scan sequential cores. The main difference lies in the fact that in case of a test-per-scan scheme, the test application is done via scan chains and one test cycle is longer than one clock cycle. This is valid both for the pseudorandom and the deterministic test. As every core contains scan chains with different lengths, the analysis procedure has to account for this and switching from one core to another has to respect the local, core-level test cycles. In the following, the experimental results with systems where every individual core is equipped with Self-Test Using MISR and Parallel Shift Register Sequence Generator (STUMPS) [6] are presented [39].

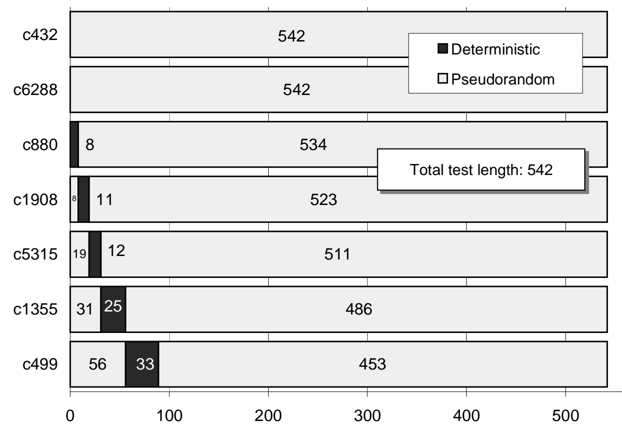



**Fig. 12.** Test schedule for the system S2 ($M_{LIMIT}$ = 5500).

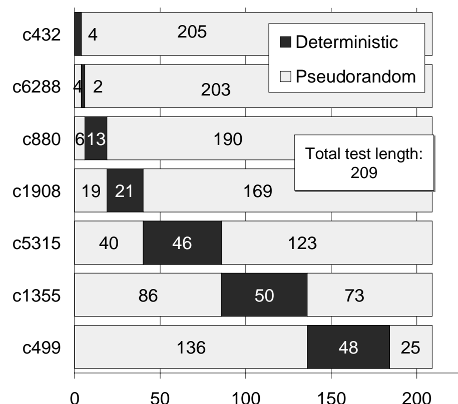



**Fig. 13.** Test schedule for the system S2 ($M_{LIMIT}$ = 14 000).

While every core has its own STUMPS architecture, at the system level we assume the same architecture as described earlier: every core's BIST logic is capable of producing a set of independent pseudorandom test patterns, i.e. the pseudorandom test sets for all the cores can be carried out simultaneously. The deterministic tests, on the other hand, can only be 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 Fig. 14.

Experiments have been performed with several systems, composed of different ISCAS'89 benchmarks as cores. All cores have been redesigned to include full scan path (one or several). The STUMPS architecture was simulated in software and for deterministic test pattern generation a commercial ATPG tool was used. The results are presented in Table 2. In Table 2 we compare our

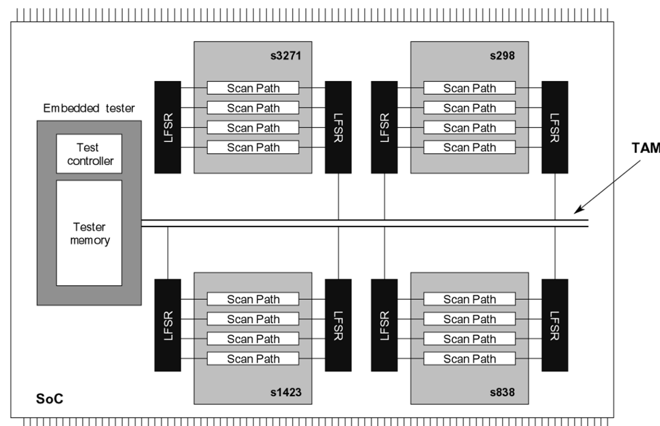



**Fig. 14.** Example of a core-based system with the STUMPS test architecture.

**Table 2.** Experimental results with STUMPS architecture

| SoC | Number of cores | Memory constraint, bits | Exhaustive approach | | Our approach | |
|---|---|---|---|---|---|---|
| | | | Total test length, clocks | CPU time*, s | Total test length, clocks | CPU time, s |
| J | 6 | 25 000 | 5750 | 57 540 | 5775 | 270 |
| | | 22 000 | 7100 | | 7150 | 216 |
| | | 19 000 | 9050 | | 9050 | 335 |
| K | 6 | 22 000 | 5225 | 53 640 | 5275 | 168 |
| | | 17 000 | 7075 | | 7075 | 150 |
| | | 13 000 | 9475 | | 9475 | 427 |
| L | 6 | 15 000 | 3564 | 58 740 | 3570 | 164 |
| | | 13 500 | 4848 | | 4863 | 294 |
| | | 12 200 | 9350 | | 9350 | 464 |

* CPU time for calculating all possible hybrid BIST solutions.

approach, where the test length is found based on estimates, with an exact approach, where deterministic test sets have been found by a brute force method (repetitive use of test pattern generator) for every possible switching point between pseudorandom and deterministic test patterns. As it can be seen from the results, our approach gives significant speedup (several orders of magnitude), while retaining very high accuracy.

## 8. CONCLUSIONS

In this paper we have presented an approach for improving the classical BIST technique, called hybrid BIST. The method is based on a hybrid test set that is composed of a limited number of pseudorandom test vectors and some additional deterministic test patterns that are specially designed to shorten the pseudo-random test cycle and to target random resistant faults.

We have described hybrid BIST cost calculation models and proposed algorithms for test time minimization, based on different test architectures. Due to the complexity of optimizing several SoC test parameters simultaneously, we have devised a solution, where one of the parameters is constrained (test memory) and we try to minimize the second one (test time). This approach is important, for example, in handheld devices where the available memory is usually very limited. The experimental results have demonstrated the efficiency of the proposed approach.

## ACKNOWLEDGEMENTS

This work has been supported by the Estonian Science Foundation (grants Nos. 6829 and 5910), Enterprise Estonia Project Technology Development Centre ELIKO, the Swedish Foundation for Strategic Research (SSF) under the Strategic Integrated Electronic Systems Research (STRINGENT) program, and the Swedish National Program on Socware (System-on-Chip).

## REFERENCES

1. *The International Technology Roadmap for Semiconductors. 2005 Edition (ITRS 2005)*. Semiconductor Industry Assoc., 2005. http://public.itrs.net/
2. Zorian, Y., Marinissen, E. J. and Dey, S. Testing embedded core-based system chips. In *Proc. IEEE International Test Conference*. Washington, 1998, 130–143.
3. Marinissen, E. J. and Zorian, Y. Challenges in testing core-based system ICs. *IEEE Communic. Mag.*, 1999, **37**, 104–109.
4. Flint, A. Multichip module self-test provides means to test at speed. *EE Eval. Eng.*, 1995, Sept., 46–55.
5. Agrawal, V. D., Kime, C. R. and Saluja, K. K. A tutorial on built-in self-test. *IEEE Design Test Comput.*, 1993, **10**, 69–77.

6. Bardell, P. H., McAnney, W. H. and Savir, J. *Built-In Test for VLSI Pseudorandom Techniques*. J. Wiley, Washington, 1987.
7. Yarmolik, V. N. and Kachan, I. V. *Self-Checking VLSI Design*. Elsevier Science, Amsterdam, 1993.
8. Golomb, S. W. *Shift Register Sequences*. Aegan Park Press, Laguna Hills, 1982.
9. Chatterjee, M. and Pradhan, D. K. A novel pattern generator for near-perfect fault-coverage. In *Proc. IEEE VLSI Test Symposium*. Princeton, 1995, 417–425.
10. Hellebrand, S., Tarnick, S., Rajski, J. and Courtois, B. Generation of vector patterns through reseeding of multiple-polynomial linear feedback shift registers. In *Proc. IEEE International Test Conference*. Baltimore, 1992, 120–129.
11. Hellebrand, S., Wunderlich, H.-J. and Hertwig, A. Mixed-mode BIST using embedded processors. *J. Electron. Test.*, 1998, **12**, 127–138.
12. Koenemann, B. LFSR-coded test patterns for scan designs. In *Proc. European Test Conference*. Munich, 1991, 237–242.
13. Sugihara, M., Date, H. and Yasuura, H. Analysis and minimization of test time in a combined BIST and external test approach. In *Proc. IEEE Design, Automation & Test in Europe Conference*. Paris, 2000, 134–140.
14. Touba, N. A. and McCluskey, E. J. Synthesis of mapping logic for generating transformed pseudo-random patterns for BIST. In *Proc. IEEE International Test Conference*. Washington, 1995, 674–682.
15. Zacharia, N., Rajski, J. and Tyzer, J. Decompression of test data using variable-length seed LFSRs. In *Proc. IEEE VLSI Test Symposium*. Princeton, 1995, 426–433.
16. Hellebrand, S., Rajski, J., Tarnick, S., Venkataraman, S. and Courtois, B. Built-in test for circuits with scan based on reseeding of linear feedback shift registers. *IEEE Trans. Comput.*, 1995, **44**, 223–233.
17. Wunderlich, H.-J. and Kiefer, G. Bit-flipping BIST. In *Proc. ACM/IEEE International Conference on CAD-96*. San Jose, 1996, 337–343.
18. Jervan, G., Peng, Z. and Ubar, R. Test cost minimization for hybrid BIST. In *Proc. IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems*. Yamanashi, 2000, 283–291.
19. Jervan, G., Peng, Z., Ubar, R. and Kruus, H. A hybrid BIST architecture and its optimization for SoC testing. In *Proc. IEEE International Symposium on Quality Electronic Design*. San Jose, 2002, 273–279.
20. Bardell, P. H. and McAnney, W. H. Self-testing of multichip logic modules. In *Proc. IEEE International Test Conference*. Philadelphia, 1982, 200–204.
21. Flynn, D. AMBA: Enabling reusable on-chip designs. *IEEE Micro*, 1997, **17**, 20–27.
22. Angelbro, C. *P-Bist Test Method Conquer the Ericsson World*. Ericsson Telecom AB, Stockholm, 1997.
23. Ubar, R., Jervan, G., Peng, Z., Orasson, E. and Raidma, R. Fast test cost calculation for hybrid BIST in digital systems. In *Proc. Euromicro Symposium on Digital Systems Design*. Warsaw, 2001, 318–325.
24. Jervan, G., Peng, Z., Ubar, R. and Korelina, O. An improved estimation methodology for hybrid BIST cost calculation. In *Proc. IEEE Norchip Conference*. Oslo, 2004, 297–300.
25. Kirkpatrick, S., Gelatt, C. D. and Vecchi, M. P. Optimization by simulated annealing. *Science*, 1983, **220**, No. 4598, 671–680.
26. Ubar, R., Kruus, H., Jervan, G. and Peng, Z. Using Tabu Search Method for optimizing the cost of hybrid BIST. In *Proc. 16th Conference on Design of Circuits and Integrated Systems*. Porto, 2001, 445–450.
27. Glover, F. Future paths for integer programming and links to artificial intelligence. *Comput. Oper. Res.*, 1986, **5**, 533–549.
28. Jervan, G., Eles, P., Peng, Z., Ubar, R. and Jenihhin, M. Test time minimization for hybrid BIST of core-based systems. In *Proc. 12th IEEE Asian Test Symposium*. Kenting, 2003, 318–323.

29. Chakrabarty, K. Test scheduling for core-based systems using mixed-integer linear programming. *IEEE Tran. Computer-Aided Des. Integr. Circ. Syst.*, 2000, **19**, 1163–1174.
30. Chou, R., Saluja, K. and Agrawal, V. Scheduling tests for VLSI systems under power constraints. *IEEE Trans. VLSI Syst.*, 1997, **5**, 175–185.
31. Garg, M., Basu, A., Wilson, T. C., Banerji, D. K. and Majithia, J. C. A new test scheduling algorithm for VLSI systems. In *Proc. CSI/IEEE Symposium on VLSI Design*, New Delhi, 1991, 148–153.
32. Larsson, E. and Peng, Z. An integrated framework for the design and optimization of SoC test solutions. *J. Electron. Test.*, 2002, **18**, 385–400.
33. Muresan, V., Wang, X., Muresan, V. and Vladutiu, M. A comparison of classical scheduling approaches in power-constrained block-test scheduling. In *Proc. IEEE International Test Conference*. Atlantic City, 2000, 882–891.
34. Zorian, Y. A distributed BIST control scheme for complex VLSI devices. In *Proc. IEEE VLSI Test Symposium*. Atlantic City, 1993, 4–9.
35. Jervan, G., Ubar, R., Shchenova, T. and Peng, Z. Energy minimization for hybrid BIST in a system-on-chip test environment. In *Proc. 10th IEEE European Test Symposium*. Tallinn, 2005, 2–7.
36. Brglez, F. and Fujiwara, H. A neutral netlist of 10 combinational benchmark circuits and a target translator in fortran. In *Proc. IEEE International Symposium on Circuits and Systems*. Kyoto, 1985, 663–698.
37. Jervan, G., Markus, A., Paomets, P., Raik, J. and Ubar, R. A CAD system for teaching digital test. In *Proc. 2nd European Workshop on Microelectronics Education*. Grenoble, 1998, 287–290.
38. *Turbo Tester Reference Manual*. Version 3.99.03, 1999. http://www.pld.ttu.ee/tt
39. Jervan, G., Eles, P., Peng, Z., Ubar, R. and Jenihhin, M. Hybrid BIST time minimization for core-based systems with STUMPS architecture. In *Proc. 18th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems*. Boston, 2003, 225–232.

# Hübriidne mitmetuumaliste süsteemide isetestimise metoodika

Gert Jervan, Raimund Ubar ja Zebo Peng

On kirjeldatud kiipsüsteemide hübriidset isetestimise metoodikat. Iga üksiku tuuma testid kombineeritakse kahest erinevast vektorite jadast: pseudojuhuslikest vektoritest, mis genereeritakse jooksvalt, ja süsteemi salvestatud, eelnevalt genereeritud deterministlikest vektoritest. Deterministlikud vektorid on loodud nii, et lühendada pseudojuhuslikku jada ja avastada vigu, mis on immuunsed juhuslike vektorite suhtes. Et võimaldada sellist testimise strateegiat, on välja töötatud mitmeid hübriidseid isetestimise arhitektuure. Kuna hübriidse isetestimise maksumust mõjutab väga palju erinevate testijadade pikkus, siis on oluline leida nende jadade optimaalne koostis. Samas ei tohi aga ohverdada testi kvaliteeti. On kirjeldatud meetodeid, mida saab kasutada pseudojuhuslike ja deterministlike jadade vahelise optimaalse kombinatsiooni leidmiseks kiipsüsteemide testimiseks. Need meetodid võimaldavad leida etteantud mälu kitsenduste juures lühima testijada. Täpsete arvutuste asemel kasutatakse kiiret kaudse hinnangu meetodit, mille tulemusel on võimalik vältida otsinguruumi täielikku uurimist ja kiirendada lahendi leidmise protsessi. Eksperimendid on näidanud väljatöötatud meetodite efektiivsust optimaalsele lähedaste tulemuste saamisel.