A Unified Method for Assembling Global Test Schedules

Albrecht P. Stroele
Institute of Computer Design and Fault Tolerance
University of Karlsruhe
D-76128 Karlsruhe, Germany

Hans-Joachim Wunderlich *
Institute of Computer Structures
University of Siegen, Hölderlinstr. 3
D-57068 Siegen, Germany

Abstract

In order to make a register transfer structure testable, it is usually divided into functional blocks that can be tested independently by various test methods. The test patterns are shifted in or generated autonomously at the inputs of each block. The test responses of a block are compacted or observed at its output register. In this paper a unified method for assembling all the single tests to a global schedule is presented. It is compatible with a variety of different test methods. The described scheduling procedures reduce the overall test time and minimize the number of internal registers that have to be made directly observable.

KEYWORDS: Built-in self-test, data path, synthesis for testability, test scheduling

1. Introduction

Integrated circuits and systems usually require a divide and conquer approach for testing. The circuit is divided into subcircuits whose inputs and outputs are easily controllable and observable, respectively, or can generate patterns and compact test responses in a self-test mode. The work presented here assumes that during high-level synthesis or during top-down design test circuitry has already been added at the register tranfer level, and the circuit has been segmented into subcircuits that are surrounded by test registers (e.g. [1, 16]). Figure 1 shows an example.

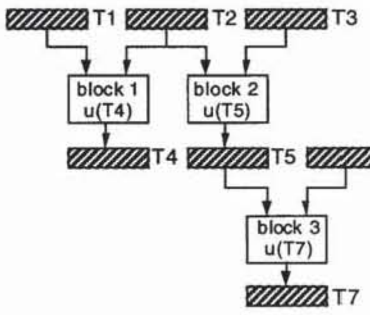

Figure 1: Example of test units at register transfer level (with test registers T_1, ..., T_7)

The test registers can be shift registers included in a (partial) scan path, or multifunctional registers for generating test patterns and compacting test responses in an autonomous mode (BIST registers). Compaction is mostly done by signature analysis whereas the generated patterns may be random (BILBO [10]), weighted random (GURT [20]), pseudo-exhaustive [2], or even deterministic patterns.

For test execution, the order of testing the subcircuits has to be specified. The test schedule should achieve a short test time, and its implementation should cause only low additional hardware cost. In this paper a scheduling approach for assembling a complete test is presented which allows combining various test strategies, e. g. (weighted) random, (pseudo-)exhaustive or deterministic, and external scan-based or built-in test.

If only a scan-based external test is considered, the test hardware overhead is independent of the test schedule. In [7, 15], a segmentation approach for this situation is described, all subcircuits are tested simultaneously. The first test session lasts until the subcircuit with the smallest test length has been tested completely. Afterwards, test patterns are applied only to the remaining subcircuits. [15] proposed a procedure to order the registers included in the scan chain such that the execution time of this test schedule is minimized.

For self-testable circuits, a different segmentation is appropriate. In order to test a portion of the circuit, at least one test register must collect test responses. Thus the smallest region that can be tested independently (*test unit*) consists of one test register that can be configured as a test response collector, the block of logic connected to the inputs of this register, and a set of test registers to supply test patterns for the inputs of the block.

In this way, every test unit $u(T_i)$ is uniquely determined by the test register T_i at its outputs. In figure 1, the test unit $u(T_4)$ includes test register T_4 (collecting test responses), logic block 1, and the test registers T_1 and T_2 (supplying patterns). The block contained in the test unit usually consists of combinational or pipeline structured logic. Test units may overlap.

* This work was supported in part by ESPRIT-project 7107 ARCHIMEDES.

In this paper, we consider block tests as defined in [5]. A block test is a test of a test unit using one specific test method. Block tests are regarded as indivisible entities for scheduling. Generally not all block tests can be performed simultaneously since they share some test resources that can be used only exclusively. If for instance deterministic patterns are applied, the block tests of test units $u(T_4)$ and $u(T_5)$ in figure 1 might have contradicting requirements on the contents of register T_2. These restrictions are described in the *test incompatibility graph* $G_I := (U, E_I)$ [5]. The nodes $u \in U$ of this graph represent the block tests, the edges connect pairs of block tests that cannot be performed simultaneously (incompatible block tests).

If a test unit is tested by multiple test methods (e.g. applying random patterns followed by some deterministic patterns), there are multiple block tests for this test unit. These block tests are pairwise incompatible and form a complete subgraph of the test incompatibility graph. For simplicity, we assume in the following that there is just one block test for each test unit. Then we do not have to distinguish between test units and their block tests explicitly. But the scheduling procedures presented in this paper can easily be extended to the general case of test units with multiple block tests.

Based on test units, the test schedule can be structured in the following way. A *test session* s_i is a set of pairwise compatible test units that are processed simultaneously. A *test schedule* $S := (s^r, E)$ is described by a sequence of test sessions $s := (s_0, s_1, \ldots, s_{d-1})$, a repetition number r, and a subset E of test registers whose contents are evaluated at test end. The set E must include all the test registers at the primary outputs since the signatures in these test registers cannot influence any other signatures. s^r is a short hand notation for the sequence where s is concatenated r times, $s^1 = s$, $s^2 = ss$, etc.

In order to minimize the number of test sessions, the set of test units must be partitioned into a minimal number of subsets with pairwise compatible test units. This problem is equivalent to coloring the nodes of the test incompatibility graph with a minimal number of colors such that no edge connects two nodes of the same color [5, 11]. The nodes with the same color represent a set of compatible test units. If for each color one test session is formed, the number of test sessions is minimum. During test execution, all the test units of a test session are started simultaneously, and the test session continues until the test unit which requires the largest test length has been finished.

The test schedule also has an impact on the number of signatures that have to be evaluated. A BIST register that operates in the signature analysis mode can get a faulty signature if the processed test unit contains a detectable fault, or if at least one of the involved pattern generating BIST registers has got a faulty signature some time before and thus produces a pattern sequence that differs from the fault-free case. In this way faulty signatures can propagate through the circuit provided that the test registers are not reinitialized during test execution [18]. If for instance in figure 1 the test of unit $u(T_5)$ produced a faulty signature in test register T_5, and afterwards this register generates patterns for $u(T_7)$, then $u(T_7)$ gets an entirely wrong pattern sequence and the responses observed in T_7 differ from the fault-free situation.

In this paper the propagation of faulty signatures is utilized to reduce the amount of self-test hardware. For many circuits the scanning of test registers and the evaluation of signatures can be restricted to a subset of the test registers and must be done only once at the end of the entire test. The advantages of this approach are short test times, a simplified self-test control and savings in hardware overhead by omitting the internal scan path.

The rest of this paper is organized as follows: The next section introduces test units more formally and discusses the compatibility of test units that are subject to different test strategies. In section 3, the resulting test incompatibility graphs are constructed and test scheduling is reduced to graph coloring. Furthermore, an algorithm is presented which builds test schedules such that no internal signatures need to be evaluated, an internal scan path is not required, and the complete test information is available at the primary outputs. Finally, in section 4 some experimental results are discussed.

2. Test units

2.1 Register transfer description

Test scheduling procedures require a description of the circuit under test including BIST and scan registers. A common description is the register graph that is produced as an intermediate structure during high-level synthesis or during top-down design (see figure 2).

The nodes of the register graph $G_R := (V_R, E_R)$ represent the registers, and there is a directed edge between two nodes if there exists a path of combinational elements between the corresponding registers. The node set V_R includes the subsets

N: Registers without any testability features

D: Directly accessible registers
(primary inputs I, primary outputs O, and registers that are integrated into a partial or complete scan path)

T: BIST registers for generating test patterns and compacting test responses in an autonomous mode

Some or all BIST registers are directly accessible. In a fully self-testable design we have $T \supset I \cup O$.

The test register graph $G_T := (V_T, E_T)$ has the node set $V_T := D \cup T$, and for each path in G_R that connects two nodes of V_T only via nodes of N there is a corresponding edge in E_T. If for instance $D = \{R_3, R_4, R_5\}$ and $T = \emptyset$ (figure 2), then the test register graph is as shown in figure 3.

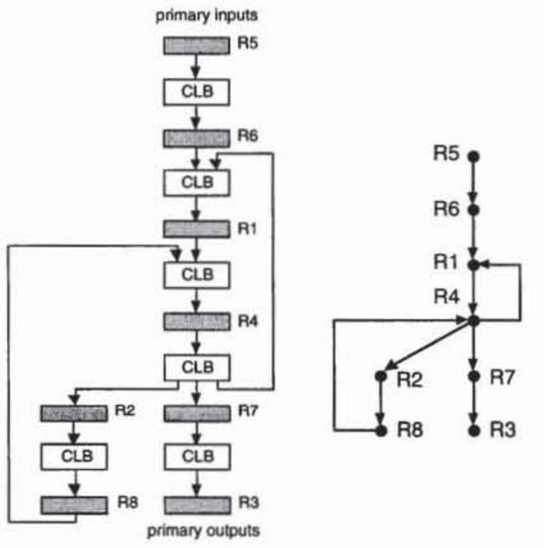

Figure 2: Register transfer structure and corresponding register graph

The hardware overhead for testability reasons depends on both the number of scannable registers and the number of BIST registers. Including BIST registers in a (partial) scan path increases the overhead as the BIST registers need an extra shift mode, additional area is required for wiring, and test control is more complex.

2.2 Placement of test registers

In order to obtain testable subcircuits, the scannable registers and the BIST registers must be placed at appropriate positions. Breaking all cycles in the circuit structure by a partial scan path bounds the length of the required test sequences to the sequential depth of the circuit [4,7,12, 14].

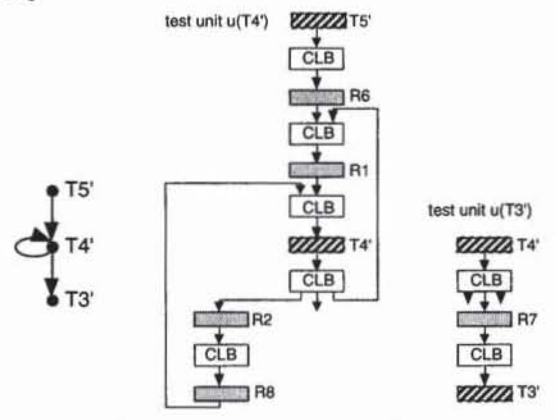

Figure 3: Test register graph and test units for scan-based testing

The placement of test registers as shown in figure 3 breaks all cycles and is suited to scan-based testing. As T_5' is not used for collecting test responses, we obtain two test units.

For scan-based testing, the output register of a test unit may also serve as one of its input registers. For BIST, however, the circuit structure obtained from breaking all cycles is not a priori suited to BIST since during self-testing some registers may have to generate patterns and compact test responses concurrently (e.g. T_4'). This kind of parallel self-test, where the signatures are used as test patterns, is feasible in some cases [9, 19], but in general the required properties of the test patterns (exhaustive, random, etc.) cannot be guaranteed. So additional test registers are required such that all cycles are broken at least twice. Hence for BIST the set T must include R_1, ..., R_5, and we obtain the test register graph and the test units shown in figure 4.

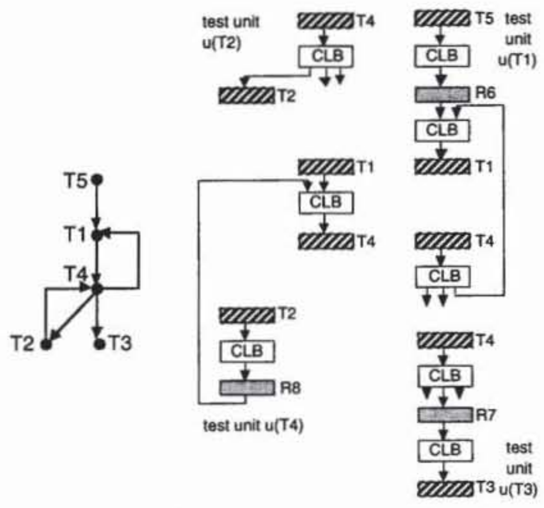

Figure 4: Test register graph and test units for BIST

2.3 Compatibility of test units

Test units can be tested by deterministic, random, weighted random or (pseudo-)exhaustive patterns. The chosen test methods have an impact on the compatibility of test units:

a) **Scan-based test** (patterns generated externally, responses evaluated externally)
   - a1: *Random patterns:* All test units can be processed concurrently as all the input patterns are equally distributed.
   - a2: *Weighted random, pseudo-exhaustive or deterministic patterns:* Two test units are compatible if their input registers are disjoint. Otherwise they are incompatible since weights, distributions, and pattern sets are specific to each

test unit. In figure 1 only the test units $u(T_4)$ and $u(T_5)$ are incompatible.

b) **BIST** (patterns generated internally, responses compacted internally)

b1: *Random patterns:* Two test units are incompatible if the output register of one is an input register of the other. In this situation the register would have to compact and generate patterns simultaneously, and random properties could not be guaranteed. In the example of figure 1, $u(T_5)$ and $u(T_7)$ are incompatible.

b2: *Weighted random, pseudo-exhaustive or deterministic patterns:* In addition to the condition of b1, here also the input registers have to be disjoint. Hence $u(T_5)$ and $u(T_7)$ are incompatible, but also $u(T_4)$ and $u(T_5)$.

Using these conditions, the test incompatibility graph $G_I := (U, E_I)$ is constructed where the edges connect incompatible test units. Depending on the test method applied, figure 5 shows the resulting test incompatibility graph for the circuit of figure 2.

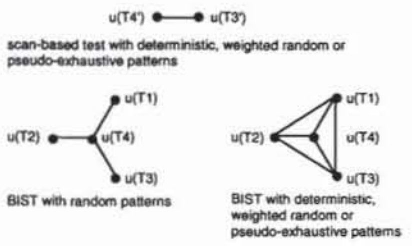
u(T4')
u(T3')
scan-based test with deterministic, weighted random or pseudo-exhaustive patterns
u(T1)
u(T2)
u(T4)
u(T3)
BIST with random patterns
u(T1)
u(T2)
u(T4)
u(T3)
BIST with deterministic, weighted random or pseudo-exhaustive patterns

Figure 5: Test incompatibility graphs for the structure of figure 2

3. Test scheduling

In this section, general test scheduling procedures are presented that can be applied to circuits with a variety of different testing techniques. Let E be the set of all the test registers that are used for collecting test responses. If the only objective is a minimum number of test sessions, then a test schedule $S = ((s_0, s_1, ..., s_{d-1}), E)$ has to be constructed where d is minimum. This problem can be solved by coloring the test incompatibility graph G_I.

In order to reduce the BIST hardware overhead, the number of evaluated signatures should be minimized. The signatures collected in the test registers at the primary outputs must always be evaluated since they cannot influence any other signatures. Usually these signatures are sufficient. Only if some test registers at the primary outputs have a relatively high aliasing probability, other test registers must be added to the minimal set of test registers, E_{min}, whose contents have to be evaluated at test end. The following problem has to be solved.

Test scheduling with a fixed (minimal) set of evaluated signatures:

Given:
- Test register graph, G_T (including aliasing probabilities of all signature analyzers)
- Set of test registers whose signatures are evaluated at test end, E_{min}
- Test incompatibility graph, G_I
- Set of faults, F
- Required fault coverage, FC_0
- Number of different test sessions, d

Find: Test schedule $S = ((s_0, s_1, ..., s_{d-1})^r, E_{min})$ where fault coverage is at least FC_0 and r is minimum

It can be shown that the corresponding decision problem contains the graph K-colorability problem [6] as a special case and hence is NP-hard.

Each fault located in a nonredundant part of a test unit $u(T_{i_1})$ can cause a faulty signature in the corresponding signature register T_{i_1}. Propagating this faulty signature along a path $(T_{i_1}, T_{i_2}, ..., T_{i_s})$ of the test register graph G_T requires the test units $u(T_{i_1}), u(T_{i_2}), ..., u(T_{i_s})$ to be processed in the same order, and the test session sequence must look like $(..., \{u(T_{i_1}), ...\}, ..., \{u(T_{i_2}), ...\}, ..., ..., \{u(T_{i_s}), ...\}, ...)$. Thus each required propagation imposes a condition on the test session sequence.

For each test register used as a signature analyzer, it is sufficient to consider a shortest path to a test register of E_{min}. There always exists such a path since the information in all nonredundant parts of the circuit has an effect on the data at the primary outputs. The resulting conditions can be summarized by a directed tree *(precedence tree)* where the nodes represent the test units and an edge $(u(T_i), u(T_j))$ means that the test unit $u(T_i)$ must be processed before the test unit $u(T_j)$.

Each node $u(T_i)$ of the tree is marked with a value $dist(u(T_i))$ that gives the length of the longest path leading to $u(T_i)$. In order to propagate the effects of all faults to the signatures at the primary outputs, at least $dist(u(T_0))$ test sessions must be executed. Figure 7 shows the precedence tree for the circuit of figure 6.

The test session sequence $(s_0, s_1, ..., s_{d-1})^r$ that satisfies all the conditions given by the test incompatibility graph and the precedence tree is built in *reversed time order.* Since the signatures at the primary outputs cannot influence any other signatures, it is best to collect them during the last test session. The signatures that can influence the signatures at the primary outputs should preferably be collected during the last but one test session, and so on.

At the beginning, all the test sessions are empty, and only the root of the precedence tree is marked as scheduled. In each step, all nodes of the precedence tree that have not yet been marked as scheduled, but whose immediate successors have already been marked as scheduled, can be included in the next test session. If not all of these candidates are compatible to each other and to the test units already included in the test session, the nodes with higher distance values are preferred. These priorities help to keep the test session sequence as short as possible. The newly included nodes are also marked as scheduled. For the example circuit of figure 6 and d=3, the resulting test schedule is $((\{u(T_3), u(T_8)\}, \{u(T_4), u(T_7)\}, \{u(T_2), u(T_5), u(T_9)\})^2, \{T_5, T_9\})$.

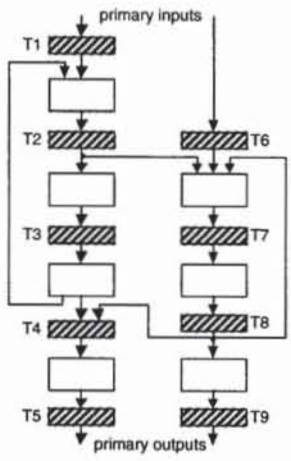

Figure 6: Example circuit

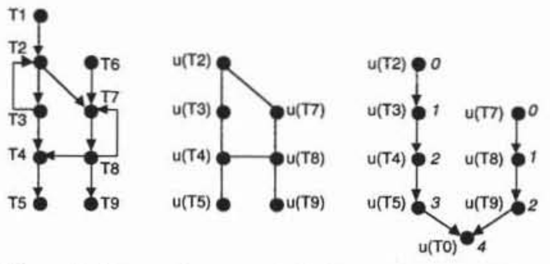

Figure 7: Test register graph (left), test incompatibility graph (center), and precedence tree (right) for the circuit of figure 6

If the constructed test schedule (s^r, E_{min}) does not achieve the required fault coverage, the number of repetitions must be increased until the value FC_0 is reached (cf. [18]). Of course a fault coverage value greater than the value obtained without test response compaction (and fault masking) cannot be obtained.

The method described above is easily extended to circuits containing some test units that are tested using a scan path. The scannable registers that collect test responses are added to the set E_{min} before the precedence tree is constructed. Then the procedure described above is applied in the same way.

4. Experimental results

This section presents the results achieved on three different designs. The circuit MU implements matrix multiplication, and its register transfer structure was generated by a high-level synthesis system [8]; the signal processor SP is the result of a manual design as described in [13]; and s35932 is one of the ISCAS'89 sequential benchmark circuits [3] whose description at register transfer level has been constructed using the method of [17]. The other sequential benchmark circuits cannot serve as examples since they seem to be control units with an irregular structure, and their descriptions at register transfer level are not known. Table 1 and table 2 list the characteristic data of the considered circuits.

circuit	# primary inputs	# primary outputs	# gates	# flip-flops
MU	43	26	879	183
SP	83	55	1354	239
s35932	35	320	16065	1728

Table 1: Gate level characteristics of the circuits

circuit	# registers (original design)	# scan registers (design with external test)	# BIST registers (design with internal test)
MU	24	14	19
SP	68	44	50
s35932	67	40	67

Table 2: Registers, scannable registers, and BIST registers

For the external scan-based test using weighted random, pseudo-exhaustive or deterministic patterns, all the registers at the primary inputs and outputs were included in a scan chain and in addition a minimal number of registers such that each cycle of the register graph contains at least one scannable register. Then the test incompatibility graphs were constructed, and the test schedules with a minimal number of test sessions were determined using the graph coloring method (table 3). Compared to sequentially testing all the test units, the number of test sessions is reduced by 50% (MU), 56% (SP) and 24% (s35932), respectively.

circuit	# test units	scheduling method	# sessions
MU	10	graph coloring	5
SP	25	graph coloring	11
s35932	37	graph coloring	28

Table 3: Test scheduling for external test

For the internal test, multifunctional test registers were placed at the primary inputs and outputs, and a minimal number of additional BIST registers was inserted such that each cycle of the register graph includes at least two of them. In general, we get other test units than those obtained in the scan designs. The test incompatibility graphs differ for random patterns on the one hand and weighted random, pseudo-exhaustive or deterministic patterns on the other hand.

Table 4 and table 5 show the scheduling results of the graph coloring method and the procedure described in section 3 (called SEQUENCE). The graph coloring method assumes that all test registers are observable, whereas SEQUENCE drives the signatures to the primary outputs and thus reduces the number of test registers that must be observable from 15 to 4 (MU), from 31 to 13 (SP), and from 64 to 10 (s35932), respectively. d is the number of different test sessions. r denotes how often these test sessions must be executed so that the faults of all test units can influence the signatures at the primary outputs. The algorithm provides various solutions where a larger d in general complicates test control, and a larger r prolongs test time.

circuit	scheduling method	# sessions (r*d)
MU (15 test units)	graph coloring	2
	SEQUENCE	3 * 2
	SEQUENCE	1 * 5
SP (31 test units)	graph coloring	3
	SEQUENCE	1 * 4
s35932 (64 test units)	graph coloring	2
	SEQUENCE	4 * 2
	SEQUENCE	1 * 7

Table 4: Test scheduling for internal test using random patterns

Table 5 shows that using weighted random, (pseudo-) exhaustive or deterministic patterns has a strong impact on self-test control and test application time as both the number of different test sessions, d, and the total number of executed test sessions, r*d, increase.

circuit	scheduling method	# sessions (r*d)
MU (15 test units)	graph coloring	5
	SEQUENCE	2 * 6
	SEQUENCE	1 * 7
SP (31 test units)	graph coloring	11
	SEQUENCE	1 * 11
s35932 (64 test units)	graph coloring	28
	SEQUENCE	1 * 29

Table 5: Test scheduling for internal test using weighted random, pseudo-exhaustive or deterministic patterns

5. Conclusions

A unified method for test scheduling is presented that takes into account combinations of various test methods as external, scan-based or built-in test, and random, weighted random, (pseudo-)exhaustive or deterministic patterns. In order to get a test schedule with a minimal number of test sessions, a graph coloring approach is applied. For BIST, the scheduling algorithm has been extended such that a short test application time (compared to testing all the test units sequentially) and reduced test hardware overhead are obtained. It drives internal signatures to the primary outputs of the circuit such that no shift mode of the BIST registers and no internal scan path are required for scanning signatures. Moreover, signatures need only be evaluated after completing the entire test, and self-test control is simplified.

6. References

[1] M. S. Abadir, M. A. Breuer: "A Knowledge-Based System for Designing Testable VLSI Chips", IEEE Design&Test, Aug. 1985, pp. 56-68

[2] Z. Barzilai, D. Coppersmith, A. L. Rosenberg: "Exhaustive Generation of Bit Patterns with Applications to VLSI Self-Testing", IEEE Trans. on Computers, Feb. 1983, pp. 190-194

[3] F. Brglez, D. Bryan, K. Kozminski: "Combinational Profiles of Sequential Benchmark Circuits", Int. Symp. on Circuits and Systems, 1989, pp 1929-1934

[4] K.-T. Cheng, V. D. Agrawal: "A Partial Scan Method for Sequential Circuits with Feedback", IEEE Trans. on Computers, April 1990, pp. 544 - 547

[5] G. L. Craig, C. R. Kime, K. K. Saluja: "Test Scheduling and Control for VLSI Built-In Self-Test", IEEE Trans. on Computers, Sept. 1988, pp. 1099-1109

[6] M. R. Garey, D. S. Johnson: "Computers and Intractability", Freeman, New York, 1979

[7] R. Gupta, R. Gupta, M. A. Breuer: "The BALLAST Methodology for Structured Partial Scan Design", IEEE Trans. on Computers, April 1990, pp. 538-544

[8] P. Gutberlet: "Entwurf eines schnellen Matrizen-multiplizierers" (in German), Studienarbeit, Department of Computer Science, University of Karlsruhe, 1988

[9] K. Kim, D. Ha, J. Tront: "On Using Signature Registers as Pseudorandom Pattern Generators in Built-in Self Testing", IEEE Trans. on CAD, 1988, pp. 919-928

[10] B. Koenemann, J. Mucha, G. Zwiehoff: "Built-In Logic Block Observation Techniques", Test Conference, Cherry Hill NJ, 1979, pp. 37-41

[11] A. Krasniewski, A. Albicki: "Automatic Design of Exhaustively Self-Testing Chips with BILBO Modules", Int. Test Conference, Washington D.C., 1985, pp. 362-371

[12] A. Kunzmann, H.-J. Wunderlich: "An analytical approach to the partial scan problem", J. of Electronic Testing: Theory and Applications, 1990, pp. 163-174

[13] J. LeBlanc: "LOCST: A Built-In Self-Test Technique", IEEE Design&Test, Nov. 1984, pp. 45-52

[14] D. H. Lee, S. M. Reddy: "On Determining Scan Flip-Flops in Partial-Scan Designs", Int. Conference on CAD, 1990, pp. 322-325

[15] S. Narayanan, C. Njinda, , M. Breuer: "Optimal Sequencing of Scan Registers", Int. Test Conference, Baltimore MD, 1992, pp. 293-302

[16] C. Papachristou, S. Chiou, H. Harmanani: "A data path synthesis method for self-testable designs", Design Automation Conference, 1991, pp. 378-384

[17] A. P. Stroele: "Partitioning and Hierarchical Description of Self-Testable Designs", IFIP Transactions, Vol. A-42 (VLSI'93), 1994, pp. 113-122

[18] A. P. Stroele, H.-J. Wunderlich: "Signature Analysis and Test Scheduling for Self-Testable Circuits", Int. Symposium on Fault-Tolerant Computing, Montreal, 1991, pp. 96-103

[19] L. T. Wang, E. J. McCluskey: "Concurrent Built-in Logic Block Observer (CBILBO)", Int. Symposium on Circuits and Systems, 1986, pp. 1054-1057

[20] H.-J. Wunderlich: "Self Test Using Unequiprobable Random Patterns", Int. Symp. on Fault-Tolerant Computing, Pittsburgh, 1987, pp. 258-263