SIGNATURE ANALYSIS AND TEST SCHEDULING FOR SELF-TESTABLE CIRCUITS

Albrecht P. Ströle, Hans-Joachim Wunderlich

University of Karlsruhe, Institute of Computer Design and Fault Tolerance (Prof. Dr. D. Schmid)
P. O. Box 6980, D-7500 Karlsruhe 1, Federal Republic of Germany

Abstract

Usually in complex circuits the test execution is divided into a number of subtasks, each producing a signature in a self-test register. These signatures influence one another. In this paper it is shown how test schedules can be constructed, in order to minimize the number of signatures to be evaluated. The error masking probabilities decrease, when the subtasks of the test execution are repeated in an appropriate order, and an equilibrium situation is reached where the error masking probabilities are minimal.

A method is presented for constructing test schedules such that only the signatures at the primary outputs must be evaluated to get a sufficient fault coverage. Then no internal scan path is required, only few signatures have to be evaluated at the end of the test execution, and the test control at chip and board level is simplified. The amount of hardware to implement a built-in self-test is reduced significantly.

KEYWORDS: Aliasing, built-in self-test, signature analysis, test scheduling.

1. Introduction

In order to implement a built-in self-test often some system registers are augmented to multi-mode *self-test registers* (STRs) like the well-known BILBO (built-in logic block observer [17]), GURT (generator of unequiprobable random tests [23]), or additive cellular automata [12]. In the test mode, STRs generate patterns or perform signature analysis. By an appropriate placement of the STRs, in the test mode all global feedback loops are cut, and the circuit is subdivided into segments (*test units*) that are completely bounded by STRs (e.g. [4, 18]).

A test unit can be tested independently from the rest of the circuit. It contains a set of STRs that generate pseudorandom or (pseudo-)exhaustive test patterns for the block under test and one STR that is configured as a multiple input signature register (MISR) to evaluate the test responses, when the test unit is processed. If the obtained signature differs from the correct signature, the block is faulty. However, even false test responses may result in the correct signature. This is called error masking or aliasing.

The test of the whole circuit consists of processing all the test units. To reduce the test execution time, one tries to process many test units concurrently. The problem of *test scheduling* is to organize the execution of all the single test units so that the available resources are optimally utilized. In order to obtain a completely self-testable circuit, the test schedule must be implemented by a test control unit (e.g. [11]).

Known algorithms for test scheduling reduce the problem to finding a minimum coloring of a graph [4, 18] (see section 2). Others aim at systems with pipeline structures and apply techniques similar to pipeline optimization techniques [1] or use special heuristics [20]. All of them try to *minimize the overall test length*. But with a built-in self-test (BIST), it is often more desirable to minimize the silicon area required for BIST structures. In this paper a scheduling method is presented that aims at *reducing the amount of additional test hardware*.

In [22] it is shown that the effects of a fault, namely faulty signatures, can propagate through the circuit, if the signatures are retained and the test registers are not reinitialized during the test execution. When the test units are processed in an appropriate order, it is sufficient to scan and evaluate only a subset of the signatures, which must include all the STRs at the primary outputs, because their signatures cannot propagate anywhere else. In this paper an algorithm is presented for constructing test schedules such that this minimal subset is sufficient for achieving a high fault coverage. This yields important advantages for testing at chip and board level:

- Only few signatures to evaluate
- No scan path for internal STRs
- Simplified BIST control unit
- Implementation of internal STRs with less hardware
- Simplified boundary-scan control:
  smaller number of test data registers to control by the TAP [13] (a test data register containing the internal STRs is not required)
- Simplified high level control:
  smaller number of instructions to send to the chip under test, smaller number of responses from the chip under test

Of course it must be ensured, that the fault coverage is unaffected by these hardware savings. Let E(N) be the expectation value of a random variable N, and let M be the set of modeled faults. The *fault coverage FC* is defined by

$$FC := \frac{1}{|M|} \cdot E(\#\text{faults detected by evaluated signatures}).$$

The fault coverage depends on the test schedule and the aliasing probabilities of the signature registers. The aliasing probabilities for single signature registers have been widely investigated, e.g. [5, 6, 14, 15, 19]. With increasing test lengths the aliasing probability asymptotically converges to the value 2^{-k}, if the signature register is based on an LFSR or a linear cellular automaton with an irreducible characteristic polynomial of degree k [6]. Several approaches are known to decrease the fault masking probability and thus increase the probability of faulty signatures in single test units [3, 25]. But they lead to a longer signature or more signatures or require more hardware for the signature collecting structures. In systems with multiple signature registers the method of [22] can be used to compute the expected fault coverage, when only a subset of signatures is evaluated at the end of the test execution.

The paper is organized as follows. Section 2 presents a model that can be used as a basis for test scheduling procedures. In section 3, it is shown, how faulty signatures influence one another, and how this can be utilized to increase the fault coverage in circuits where only few signatures are evaluated. Section 4 proves that the repeated processing of test units leads to an equilibrium situation where the probabilities of fault signatures in the STRs are maximal. The problem of scheduling the test execution, such that the maximal probabilities of the equilibrium are reached, is stated formally in section 5, and an algorithm for its solution is presented. Section 6 demonstrates the test scheduling methods with examples and gives a short discussion. A summary in section 7 concludes the paper.

2. Modeling and Formal Description

There is a one-to-one correspondence between the test units and the STRs that can be used as signature registers. Each test unit u_i comprises exactly one STR T_i that is configured as a signature register, and each STR T_i, that can be used as a signature register, is used in exactly one test unit u_i for signature analysis. This correspondence is indicated by the same indices (see table 1).

The effects of a fault can only propagate in the direction of the data flow, and thus the propagation depends on the circuit structure. The circuit is modeled by the *test register graph* $G_T := (T, E_T)$. Each node $T_i \in T$ represents an STR. The test register graph contains an edge $(T_g, T_i) \in E_T$ for every STR T_g that generates patterns in test unit u_i and thus influences the signature register T_i. The test register graph is independent from the test schedule. The paths of G_T describe how the effects of a fault, namely faulty signatures, can propagate *(propagation paths)*, provided an appropriate test schedule is executed. If a fault F is located in the test unit u_i, only the STR T_i and the set $s(T_i)$ of its successors can get contents that differ from the fault-free circuit.

In the following we explain all the concepts of this paper with the help of the simple example circuit of figure 1. The corresponding test register graph is shown in figure 2. The fault F is located in the test units u_1, u_2, and u_3, and can cause faulty signatures in the STRs T_1, T_2, T_3, T_4.

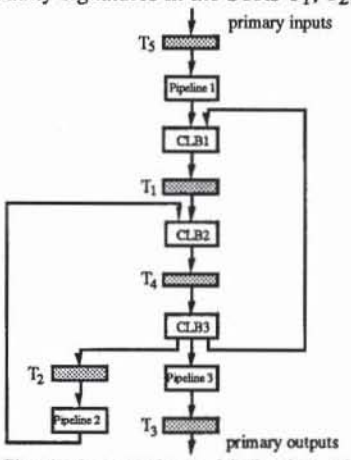

Figure 1: Circuit for matrix multiplication with built-in self-test registers T_i

When the circuit is tested, some test units can be processed in parallel. These test units are called *compatible*. On the other hand, two test units are *incompatible*, if their test hardware requirements are contradictory. For example two test units are not allowed to simultaneously send data on the same bus lines. The contents of a test register that is used as an MISR in one test unit must not be used at the same time as test patterns for another test unit (see below). These and other restrictions due to the limited test resources are modeled by the *test incompatibility graph* $G_I = (U, E_I)$ [4]. The nodes $u_i \in U$ of this graph represent the test units. The test incompatibility graph contains an edge $(u_i, u_j) \in E_I$ if and only if the test units u_i and u_j are incompatible. Figure 2 shows the test incompatibility graph for the circuit of figure 1.

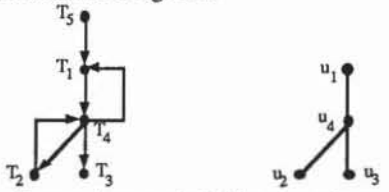

Figure 2: Test register graph (left) and test incompatibility graph (right) for the circuit of figure 1

The graph coloring approach of [4] tries to color the nodes of the test incompatibility graph with a minimal number of colors, such that no two nodes that are connected by an edge get the same color. Then all test units that correspond to nodes with the same color can be processed in parallel without violating any compatibility constraints. This directly corresponds to a test schedule represented by a sequence of test sessions $S := (S_0, S_1, ..., S_{d-1})$. A *test session* S_i is a subset of pairwise compatible test units which are processed in parallel. The test unit with the largest test length determines the test length of the test session. A test schedule for the circuit of figure 1 is $(\{u_1, u_2, u_3\}, \{u_4\})$. The authors of [4] call this the "nonpartitioned testing" method. Compared with other, more flexible test scheduling methods discussed by these authors, the test session method needs less hardware for implementing a BIST control unit.

In the subsequent sections we make the following assumptions:

a) All STRs are initialized correctly before test execution begins. (The extension to the general case is straightforward.)
b) Signatures are read out only at the end of the test execution.
c) When an STR is operating in pattern generation mode, it produces pseudorandom or pseudo-exhaustive patterns, and its contents are not affected by the incoming data sequence.
d) When an STR is operating in signature analysis mode, its contents are not at the same time used as test patterns.
e) The test lengths for each test unit are long enough to reach the stationary aliasing probabilities of the signature register.
f) When the processing of a test unit is repeated, the resulting signatures are statistically independent.

The effects of a fault can propagate through the circuit if the signatures are scanned only at the end, and if the STRs are not clocked in the normal mode or reinitialized after signature analysis. The conditions c) and d) ensure that the generated patterns do not depend on the circuit function and are really pseudorandom or pseudo-exhaustive. In some cases sufficient fault coverage may be obtained without these assumptions [7, 16], but the known methods for computing aliasing probabilities [5, 6, 14] do not hold if an STR is part of a feedback loop. Generally condition e) holds as the test lengths increase with the reciprocal of the lowest detection probability [21, 24]. The calculations and the simulation results of [6, 14] show, that the aliasing probability converges fast to its stationary value, when a linear feedback shift register (LFSR) with a primitive feedback polynomial is used. In the stationary situation all states of the MISR (width k) occur with the same probability 2^{-k}, and this is independent from the initial state of the MISR. The assumption f) holds if the starting values of the pseudorandom or pseudo-exhaustive pattern generators are statistically independent.

3. Mutual Influence of Faulty Signatures

In circuits with multiple signature registers, the signatures can influence one another. Figure 3 and 4 illustrate this with an example. The circuit is segmented into two test units (table 1).

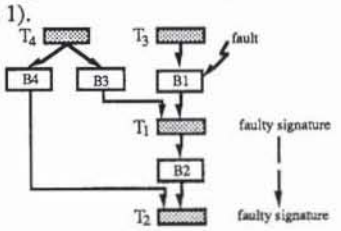

Figure 3: Propagation of faulty signatures

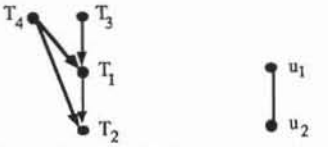

Figure 4: Test register graph (left) and test incompatibility graph (right) of the circuit of figure 3

Test unit	pattern generating test registers	blocks under test	signature register
u_1	T_3, T_4	B_1, B_3	T_1 (width k_1)
u_2	T_1, T_4	B_2, B_4	T_2 (width k_2)

Table 1: Test units for the circuit of figure 1

The combinational logic block B_1 contains a detectable fault (fault detection probability 1). The test schedule is $(\{u_1\}, \{u_2\})$. Provided that error masking does not occur, T_1 will contain a faulty signature after the test unit u_1 was processed. The STRs are not initialized again. Due to the incorrect starting value, the test pattern sequence produced by T_1 during the processing of the test unit u_2 will not be the same as in the fault-free case, and with a high probability the signature in T_2 will also get faulty. The probabilities of a faulty signature in T_1 and T_2 at test end are $1-2^{-k_1}$ and $(1-2^{-k_1})\cdot(1-2^{-k_2})$, respectively. Generally a decreasing probability of faulty signatures is observed in the STRs along the propagation paths of faulty signatures.

In this way a faulty signature, that occurs during the test execution, may cause further faulty signatures, even in parts of the circuit that themselves are fault-free. The propagation of faulty signatures depends on the order in that the test units are processed. A correctly initialized STR can get an incorrect signature, only if it is operating as a signature register and the processed test unit contains

a detectable fault, or if at least one of the involved pattern generating STRs has got an incorrect starting value and consequently produces a different pattern sequence. Any difference between the actual contents of an STR and the contents corresponding to the fault-free circuit will remain unchanged in the pattern generation mode.

Since in the presented approach the signatures are read only at the end of the test execution, processing test units repeatedly can also increase the probabilities of faulty signatures. The effect of repeating test units is modeled by boolean variables:

$b_i^{(j)}$: STR T_i contains a faulty signature after the j-th repetition

$c_i^{(j)}$: there is no aliasing in T_i during the j-th repetition

Let $O := \{T_{i_1}, ..., T_{i_m}\}$ be the set of STRs whose signature is evaluated (set of the STRs at the primary outputs). Then $P(b_{i_1}^{(j)} \vee ... \vee b_{i_m}^{(j)})$ is the probability that at least one of these STRs contains a faulty signature. The term can be computed exactly as shown in [22].

For the example of figure 3 and the repeated processing of the test units u_1 and u_2, the propagation can be described by the boolean equations

$b_1^{(j)} = c_1^{(j)}$ and $b_2^{(j)} = (b_1^{(j)} \wedge c_2^{(j)}) \vee (\neg b_1^{(j)} \wedge b_2^{(j-1)})$.

The second term is responsible for the increasing probabilities of a faulty signature in T_2. Table 2 demonstrates this effect. $(\{u_1\},\{u_2\})^r$ is used as a short hand notation for the test schedule where $(\{u_1\},\{u_2\})$ is concatenated r times, e.g. $(\{u_1\},\{u_2\})^2 = (\{u_1\},\{u_2\}, \{u_1\},\{u_2\})$. Each repetition increases the probability of a faulty signature. If the STRs T_1 and T_2 are 8 bit wide, the maximum probability for a faulty signature in T_2 is (almost exactly) reached after 3 repetitions. With 16 bit wide STRs only 2 repetitions are needed. A detailed analysis of the maximum probabilities, that can be obtained in the general case, follows in the next section.

Test schedule	probability of a faulty signature in STR T_2 at test end	
	8 bit STRs	16 bit STRs
$(\{u_1\},\{u_2\})$	0.992203	0.999970
$(\{u_1\},\{u_2\})^2$	0.996079	0.999985
$(\{u_1\},\{u_2\})^3$	0.996094	0.999985
$(\{u_1\},\{u_2\})^\infty$	0.996094	0.999985

Table 2: Probabilities of faulty signatures

4. The Equilibrium Distribution for the Probabilities of Faulty Signatures

In this section it is shown that after a number of repetitions a stationary equilibrium situation is reached where the probabilities that an STR contains a faulty signature are maximum. It is assumed that the test schedule $(S_0, S_1, ..., S_{d-1}, S_0, S_1, ..., S_{d-1}, ...)$ is a periodically repeated fixed sequence of d test sessions, that contain each test unit at least once.

The contents of an STR T_i with width k_i can be described by a bit vector $\mathbf{b} = (b_{k_i-1}, ..., b_0)$, $b_j \in \{0, 1\}$ for $j = 0, 1, ..., k_i-1$. Since signature analysis is a linear operation, only the error vector $\mathbf{e} = \mathbf{b'} \oplus \mathbf{b}$, the difference of the correct contents $\mathbf{b'}$ (corresponding to the fault-free circuit) and the actual contents $\mathbf{b}$, has to be considered in order to determine aliasing [6]. $\oplus$ is the sum in GF(2) for each component of the vector. To simplify the notation, the values of this error vector $\mathbf{e}$ are coded by the integers $0, 1, ..., 2^{k_i}-1$, where the integer 0 corresponds to the vector $\mathbf{0}$, that indicates the correct signature.

The error vectors of all STRs describe the faulty signatures in the circuit. The propagation process can be described by a *Markov chain MC:*

State: $\mathbf{z}(r) := (z_1(r), z_2(r), ..., z_n(r))$
The component $z_i(r)$ codes the error vector $\mathbf{e}$ of the STR T_i after the r-th repetition of the test session sequence, $i = 1, ..., n$.

State space: $Z := \{0, 1, ..., 2^{k_1}-1\} \times ... \times \{0, 1, ..., 2^{k_n}-1\}$
where n: # STRs
k_i: width of STR T_i, $i = 1, 2, ..., n$

State transition: Change during one repetition of the test session sequence

Initial distribution: $\mathbf{z}(0) = \mathbf{0}$
(all STRs initialized correctly)

Theorem:
If the test schedule is a periodically repeated fixed sequence of test sessions, that contain each test unit at least once, the probability that an STR T_i contains a faulty signature after a large number r of repetitions of the test session sequence is

$$\lim_{r\to\infty} P(z_i(r) \neq 0) = \begin{cases} 0 & \text{if } T_i \notin T_F \cup s(T_F) \\ 1 - 2^{-k_i} & \text{if } T_i \in T_F \cup s(T_F) \end{cases}$$

where k_i: width of the STR T_i
$T_F \subset T$: set of STRs that are used as signature registers in the test units where the fault F is located
$s(T_F)$: set of successors of the nodes of T_F in the test register graph

Proof:
The Markov chain MC is homogeneous for any given fault. For a specific fault F in test unit u_i not all states $z \in Z$ are possible, since faulty signatures can influence other signatures only in the direction of the data flow. Only those components $z_j(r)$ of $\mathbf{z}(r)$ can change where

$T_j \in T_F \cup s(T_F)$. All the other components $z_j(r)$ are always 0, $P(z_j(r) \neq 0) \equiv 0$.

Using $T' := T_F \cup s(T_F) = \{T_{i_1}, T_{i_2}, ..., T_{i_{n'}}\}$, $n' \leq n$, only the reduced state $z_F(r) := (z_{i_1}(r), z_{i_2}(r), ..., z_{i_{n'}}(r))$ and the corresponding transition probability matrix P_F are of interest. Every state of the reduced state space $Z_F := \{0, 1, ..., 2^{k_{i1}} - 1\} \times ... \times \{0, 1, ..., 2^{k_{in'}} - 1\}$ can be reached from every other state. Starting from an arbitrary state, the system can go to a state where all STRs of the set T' contain a faulty signature. In this situation bit errors can occur at the inputs of every signature register of T', and the signatures can get arbitrary values. Hence the Markov chain MC_F with state space Z_F and transition probability matrix P_F is irreducible. Since every repetition of the test session sequence can leave the state unchanged, MC_F is also aperiodic. Consequently MC_F is ergodic [8] and for $r \to \infty$ a unique equilibrium distribution exists [9], that is stationary, too.

Let $x \in Z_F$ be the starting state of one repetition, and let $d_i(x)$ be the probability that there are bit errors at the inputs of the STR T_i during the repetition. $d_i(x)$ does not depend on r. In the case of bit errors, all contents of the STR T_i have the same probability 2^{-k_i}.

The probability that the actual contents $z_i(r)$ of an STR T_i differ from the fault-free case by x_i is

$P(z_i(r)=x_i) = [1 - d_i(z(r-1))] \cdot P(z_i(r-1)=x_i) + d_i(z(r-1)) \cdot 2^{-k_i}$

and in the equilibrium case

$P(z_i=x_i) = [1 - d_i(z)] \cdot P(z_i=x_i) + d_i(z) \cdot 2^{-k_i}$,

hence $d_i(z) \cdot P(z_i=x_i) = d_i(z) \cdot 2^{-k_i}$.

If $T_i \notin T'$, then $z_i(r) \equiv 0$, $P(z_i \neq 0) \equiv 0$. If $T_i \in T'$, then after some initial repetitions $d_i(z) \neq 0$, and consequently

$P(z_i=x_i) = 2^{-k_i}$, $P(z_i \neq x_i) = 1 - 2^{-k_i}$. ∎

For long test lengths the probability of a faulty signature in each STR T_i approaches a maximum value, that is determined only by the width k_i of the STR. It depends neither on the length of the propagation path from the location of the fault to the STR T_i, nor on the characteristic features of other STRs involved in the propagation process.

Since the information in all nonredundant parts of the circuit eventually affects the outputs of the circuit, all faults can cause faulty signatures in the STRs at the primary outputs. Thus it is sufficient to *evaluate the signatures in the STRs at the primary outputs only*, if these are not extremely small. The results in section 6 will show that the test session sequence must be repeated only few times in order to practically reach the maximum probabilities of faulty signatures given by the theorem.

5. Test Scheduling

In order to get an inexpensive implementation of the BIST control unit, the test schedule should be composed of a short sequence of test sessions, that is concatenated repeatedly. Then only the few test sessions of the short sequence must be stored, the repetitions can be implemented simply by a small counter circuit. For a given circuit and a fault set M, the fault coverage FC(O,S) obtained by evaluating the signatures in the subset O of STRs at the test end, depends on O and the test schedule S. In this context the test scheduling problem can be stated as follows.

Problem *Test Scheduling*

Given:
- Test register graph $G_T = (T, E_T)$
- Subset $O \subset T$ of test registers for signature evaluation
- Test incompatibility graph $G_I = (U, E_I)$
- Set M of faults
- Required fault coverage FC_0

Find: A test schedule $S := (S_0, S_1, ..., S_{rd-1})$
where $S_i \subset U$ and $i \equiv j \bmod d \rightarrow S_i = S_j$
for all $i, j \in \{0, 1, ..., r \cdot d - 1\}$
(d test sessions are repeated r times),
such that
1) for all $u_g, u_h \in U$, $S_i \in \{S_0, S_1, ..., S_{rd-1}\}$:
$u_g \in S_i \wedge u_h \in S_i \rightarrow \{u_g, u_h\} \notin E_I$
2) $FC(O,S) \geq FC_0$
3) d is minimal
4) r is as small as possible for minimal d

The decision problem corresponding to this optimization problem is NP-hard. It contains the graph K-colorability problem [10] as a special case. For O=T and a sufficiently high required fault coverage FC_0, all test units must be processed once (r=1). They have to be scheduled in a minimal number of test sessions. This is equivalent to coloring the nodes of the graph G_I using a minimal number of colors, each color corresponds to a test session.

The schedule must guarantee that all (nonredundant) faults can influence the signatures in the subset O of STRs. Each fault located in a test unit u_{i_1} can cause a faulty signature in the corresponding signature register T_{i_1}. When this STR T_{i_1} is used afterwards to generate patterns for a test unit u_{i_2}, the signature in STR T_{i_2} can also become faulty. The propagation of a faulty signature corresponds to travelling along a propagation path $(T_{i_1}, T_{i_2}, ..., T_{i_s})$ in the test register graph G_T. This propagation is possible only if the corresponding test units $u_{i_1}, u_{i_2}, ..., u_{i_s}$ are processed in the same order.

The test schedule must contain a test session comprising the test unit u_{i_1}, then a test session comprising u_{i_2}, and

so on. Other test sessions are allowed between these test sessions. Hence the test schedule must look like

(..., $\{u_{i_1}, ...\}$, ..., $\{u_{i_2}, ...\}$, ..., ..., $\{u_{i_s}, ...\}$, ...) .

A test schedule can be constructed in two steps:

i) A set of propagation paths is created that contains at least one path from each STR used as a signature register to an STR of O.
ii) The test sessions are built such that for each propagation path of this set the corresponding test units appear in the sequence of test sessions in the same order.

Heuristics help to choose an efficient set of propagation paths. From each signature register a *shortest* propagation path to each STR of O is selected. On the shortest propagation path the smallest number of signature registers is involved. As in each of them fault masking is possible, the fault masking probability is often lowest on the shortest path. The shortest paths also contribute to a relatively short overall test length, since in order to propagate a faulty signature along the shortest path the smallest number of test sessions is required.

Propagation paths *to all the STRs of O that can be reached* are selected, since for all subsets $O'' \subset O' \subset O$ of STRs the inequation

P (faulty signature in at least one STR of O'')
$\leq$ P (faulty signature in at least one STR of O')

holds. The set of propagation paths determined in step i) can be reduced, since a path $(T_{i_1}, ..., T_{i_j}, ..., T_{i_s})$ implies also the propagation along the path $(T_{i_j}, ..., T_{i_s})$. Hence only the propagation path $(T_{i_1}, ..., T_{i_s})$ needs to be considered, when the test sessions are constructed.

The last test session is constructed first, the scheduling is done backwards. As all propagation paths end at the STRs of O, many paths have a common subpath at the end. When the last nodes of the propagation paths are considered first, these common subpaths can be treated together, the corresponding test units must be scheduled only once, and conflicts during the scheduling process are avoided.

The set of propagation paths contains paths of different lengths. The test units that contribute to the propagation along the paths with a larger number of nodes are treated with higher priority, since these propagation paths have greater influence on the resulting overall test length than paths with a smaller number of nodes. This heuristic is not necessary to get a minimal number d of different test sessions, but it helps to avoid excessively large test lengths.

These heuristics are applied in the algorithm STS ("Self-Test Scheduling") of figure 5. The inputs are the test register graph G_T, the subset O of STRs whose signatures are evaluated, the test incompatibility graph G_I, and the required number d_{min} of test sessions. The set O can be an arbitrary subset of STRs, but the most interesting case is to include only the STRs at the primary outputs. The algorithm STS first tries to construct a sequence of $d=d_{min}$ test sessions. This is impossible, if d_{min} is less than the chromatic number $\chi(G_I)$ of the test incompatibility graph G_I. If the algorithm does not find such a sequence, the parameter d is incremented until the construction is successful. The output is the number d and the test session sequence $S = (S_0, S_1, ..., S_{d-1})$. If the sequence S does not give sufficient fault coverage, the fault masking probabilities along the propagation paths are reduced by repeating the sequence. The required number r of repetitions can be determined by computing the fault coverage values for some small numbers of repetitions. The complete test schedule is $(S_0, S_1, ..., S_{d-1}, ..., S_{rd-d}, S_{rd-d+1}, ..., S_{rd-1})$, where $S_i = S_{d+i} = ... = S_{(r-1)d+i}$ for $i = 0, 1, ..., d-1$.

```
Procedure  STS (G_T, O, G_I, d_min, d, S);
/* M:  set of propagation paths,   d: # test sessions      */
/* j:  index of currently constructed test session         */
d := d_min - 1;  M := Ø;

for each STR T_i ∈ T that is used as a signature register:
  for each STR T_o ∈ O that can be reached:
    {  determine a shortest path ω(T_i, T_o) in G_T;
       M := M ∪ {ω(T_i, T_o)}
    }
remove all paths from M that are contained in other paths
  of M;
do { for j := 0, 1, ..., d:
       S_j := Ø;
     j := d;  d := d+1;
     do { M' := Ø;
          do { L := {u_i ∈ U | ω(T_j,T_i) is a path of M with
                                  maximum number of nodes};
                                        /* set of candidates */
               choose a maximal subset L' ⊂ L, such that all
               test units of L' are compatible to each other
               and to all test units of S_j;
               S_j := S_j ∪ L';
               for all paths ω(T_j,T_i) ∈ M where u_i ∈ L:
                 { if u_i ∈ L':  delete the last node of ω;
                                 /* shorter propagation path */
                   M' := M' ∪ {ω};  M := M \ {ω}
                 }
             } while (M ≠ Ø);
          M := M';  j := (j-1) mod d;
          remove all paths from M that are contained in
            other paths of M
        } while (M ≠ Ø and during the last d loops at least
                                  one path has been shortened)
  } while (M ≠ Ø)
end;
```

Figure 5: Algorithm STS

The algorithm STS is demonstrated using the circuit of figure 1 and the corresponding test register graph $G_T = (T, E_T)$ of figure 2. Only the signature of the STR T_3 is evaluated, $O = \{T_3\}$. This is advantageous, since the STR T_3 is located at the primary outputs and can be read using the boundary scan chain. The set of shortest paths in G_T from a signature register to the STR of O is $M := \{(T_1, T_4, T_3), (T_2, T_4, T_3), (T_4, T_3), (T_3)\}$. The STR T_5 is not considered, since it is not used for signature analysis. All propagation paths that are contained in other paths of M can be removed, $M := \{(T_1, T_4, T_3), (T_2, T_4, T_3)\}$.

i) For $d_{min} = 2$ the constructed test session sequence is $(\{u_4\}, \{u_1, u_2, u_3\})$ with d=2. This is a sequence of minimal length, since $\chi(G_I) = 2$. The sequence must be repeated in order to make the propagation from all signature registers to the STR of O possible. The final test schedule is $(\{u_4\}, \{u_1, u_2, u_3\}, \{u_4\}, \{u_1, u_2, u_3\})$.

ii) For $d_{min} = 3$ the constructed test schedule is $(\{u_1, u_2\}, \{u_4\}, \{u_3\})$ with d=3.

In case ii) one more test session must be stored in the BIST control unit, but the overall test length is shorter. Generally, larger values for d_{min} lead to a larger number of test sessions to be stored, but decrease the test length. So there is a tradeoff between a smaller test hardware overhead and a shorter test length. The presented algorithm to solve the NP-hard scheduling problem cannot guarantee, that in all cases an optimal solution is found. But the constructed schedules are very close to optimal schedules as the results show.

6. Results

In this section the results of the algorithm STS are compared with the results of the "nonpartitioned testing" algorithm (NT) of [4], that uses the graph coloring approach and aims at a minimal overall test length. First we discuss two examples in more detail, after that we present results obtained with the ISCAS'89 benchmark circuits [2]. The first example E1 is the circuit of figure 1 with 20-bit-STRs, the second example E2 is the same structure but with 8-bit-STRs.

In table 3 the schedules obtained by the graph coloring algorithm of [4] and by our approach are compared. The number of repetitions is chosen such that in all cases the fault coverage is the same for both scheduling methods. For E1 the error masking probabilities are very low and can be neglected, when the algorithm STS is applied. At the end of the test execution the probability of a faulty signature in the STR T_3 differs from the equilibrium value by less than $2 \cdot 10^{-6}$. This means, the probability that a faulty signature is masked during the propagation to T_3 is less than $2 \cdot 10^{-6}$. If the STRs are 8 bit wide (E2), the probability of a faulty signature decreases along the propagation paths by an amount that cannot be neglected. This must be compensated by repeating the test session sequence once more.

circuit	#test sessions		total lengthof signatures	
	NT	STS	NT	STS
E1	2	2x2	80 bit	20 bit
E2	2	3x2	32 bit	8 bit

Table 3: Test schedules for the example circuits

In the schedules constructed by the algorithm STS, the test lengths are increased by a factor of 2 and 3, respectively, compared to the "nonpartitioned testing" schedule. But the advantage is that for E1 and E2 only the signature in the STR T_3 at the primary outputs must be evaluated at the end of the test, whereas with the "nonpartitioned testing" schedule all the signatures of T_1, T_2, T_3, and T_4 must be scanned and evaluated. There is no need for an internal scan path, the wiring required to interconnect all the test registers can be saved. The BIST control unit does not have to scan the signatures in the internal STRs and can be simplified. The control unit of the boundary-scan architecture does not need any instructions to control an internal test data register. The comparison of signatures requires less effort. Altogether the amount of test hardware is reduced significantly, the savings are quantified in table 3. Here the number of bits of the signatures derived by the graph coloring algorithm and by STS are listed. The signatures of the STS method are significantly shorter without any loss of fault coverage.

The large ISCAS'89 benchmark circuits [2] were also investigated. At all primary inputs and primary outputs, boundary scan cells were added. Then test registers were built in, such that all global feedback loops of the circuits were cut by at least two test registers. In some cases this required additional flipflops. Table 4 shows the test scheduling results.

circuit	#gates	#STRs	#test sessions		total length of signatures (bit)	
			NT	STS	NT	STS
s5378	2779	8	7	10	143	49
s9234	5597	4	4	6	335	22
s13207	7951	5	5	6	821	121
s15850	9772	3	3	5	979	87
s35932	16065	5	5	6	1017	320
s38417	22179	4	4	7	2262	106
s38584	19253	6	6	9	2567	278

Table 4: Results for ISCAS'89 benchmark circuits

Generally the schedules obtained by the algorithm STS are longer than schedules, that aim at a minimal test execution time. But the advantages of the presented approach are the hardware savings at many points when a built-in self-test is implemented. If some internal STRs are very small (e.g. less than 6 bit), then propagation paths that contain these STRs have to be repeated many

times. This can lead to a long overall test time. Very small STRs at the primary outputs can also cause problems to get a sufficient fault coverage. But in most circuits the STRs are sufficiently wide and do not cause any trouble. In most cases 2 to 4 repetitions are sufficient to reach the equilibrium and the maximum fault coverage. The detailed investigation of the tradeoff between fault coverage, test length, and amount of additional test hardware will be a subject of further research.

7. Conclusions

In large circuits with multiple signature registers, the signatures influence one another and can propagate through the circuit. The probabilities of a faulty signature decrease along the propagation path, but the repeated processing of test units can counteract this effect. After a number of repetitions a stationary equilibrium situation is reached, where for each self-test register the probability of a faulty signature is maximum and depends only on the width of this self-test register.

A method was presented to construct schedules, such that this equilibrium situation is reached and only the signatures in the self-test registers at the primary outputs must be evaluated to get a low error masking probability and sufficient fault coverage. Then a built-in self-test can be implemented with a substantially reduced amount of hardware. Only a small number of signatures must be evaluated, the internal self-test registers do not have to be integrated into an internal scan path, and the BIST control unit and the boundary-scan controller can be simplified.

References

[1] M. S. Abadir, M. A. Breuer, "Constructing Optimal Test Schedules for VLSI Circuits Having Built-In Test Hardware", Proc. International Symposium on Fault-Tolerant Computing FTCS-15, 1985, pp 165-170

[2] F. Brglez, D. Bryan, K. Kozminski, "Combinational Profiles of Sequential Benchmark Circuits", Proc. International Symposium on Circuits and Systems ISCAS'89, Portland, Oregon 1989, pp 1929-1934

[3] D. K. Bhavsar, B. Krishnamurthy, "Can We Eliminate Fault Escape in Self Testing Polynomial Division (Signature Analysis)?", Proc. International Test Conference ITC-84, 1984, pp 134-139

[4] G. L. Craig, C. R. Kime, K. K. Saluja, "Test Scheduling and Control for VLSI Built-In Self-Test", IEEE Transactions on Computers, Vol. 37, No. 9, September 1988, pp 1099-1109

[5] M. Damiani et al., "Aliasing in Signature Analysis Testing with Multiple-Input Shift-Registers", Proc. 1st European Test Conference ETC-89, Paris 1989, pp 346-353

[6] W. Daehn, T. W. Williams, K. D. Wagner, "Aliasing errors in linear automata used as multiple-input signature analyzers", IBM Journal of Research and Development, Vol. 34, No. 2/3, March/May 1990, pp 363-380

[7] B. Eschermann, H.-J. Wunderlich, "Parallel Self-Test and the Synthesis of Control Units", Proc. 2nd European Test Conference ETC-91, München 1991

[8] W. Feller, "An Introduction to Probability Theory and Its Application", Wiley & Sons, New York 1968

[9] F. Ferschl, "Markov-Ketten", Springer, Berlin 1970

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

[11] O. F. Haberl, H.-J. Wunderlich, "The Synthesis of Self-Test Control Logic", Proc. COMPEURO-89, Hamburg 1989, pp 5.134-5.136

[12] P. D. Hortensius et al., "Cellular Automata-Based Pseudorandom Number Generators for Built-In Self-Test", IEEE Transactions on CAD, Vol. 8, No. 8, August 1989, pp 842-859

[13] IEEE Standard Test Access Port and Boundary-Scan Architecture, IEEE Std 1149.1-1990, May 1990

[14] A. Ivanov, V. K. Agarwal, "An Iterative Technique for Calculating Aliasing Probability of Linear Feedback Signature Registers", Proc. International Symposium on Fault-Tolerant Computing FTCS-18, Tokyo 1988, pp 70-75

[15] K. Iwasaki, F. Arakawa, "An Analysis of the Aliasing Probability of Multiple-Input Signature Registers in the Case of a 2^m-ary Symmetric Channel", IEEE Transactions on Computer-Aided Design, Vol. 9, No. 4, April 1990, pp 427-438

[16] K. Kim, D. S. Ha, J. G. Tront, "On Using Signature Registers as Pseudorandom Pattern Generators in Built-In Self-Testing", IEEE Transactions on CAD, Vol. 7, No. 8, August 1988, pp 919-928

[17] B. Koenemann, J. Mucha, G. Zwiehoff, "Built-In Logic Block Observation Techniques", Proc. IEEE Test Conference, Cherry Hill, New Jersey 1979, pp 37-41

[18] A. Krasniewski, A. Albicki, "Automatic Design of Exhaustively Self-Testing Chips with BILBO Modules", Proc. International Test Conference ITC-85, 1985, pp 362-371

[19] D. K. Pradhan, S. K. Gupta, M. K. Karpovsky, "Aliasing Probability for Multiple Input Signature Analyzer", IEEE Transactions on Computers, Vol. 39, No. 4, April 1990, pp 586-591

[20] J. Sayah, C. R. Kime, "Test Scheduling for High Performance VLSI System Implementations", Proc. International Test Conference ITC-88, 1988, pp 421-430

[21] J. J. Shedletsky, "Random Testing: Practicality vs. Verified Effectiveness", Proc. International Symposium on Fault-Tolerant Computing FTCS-7, Los Angeles 1977, pp 175-179

[22] A. P. Ströle, H.-J. Wunderlich, "Error Masking in Self-Testable Circuits", Proc. International Test Conference ITC-90, Washington 1990, pp 544-552

[23] H.-J. Wunderlich, "Self-Test Using Unequiprobable Random Patterns", Proc. International Symposium on Fault-Tolerant Computing FTCS-17, Pittsburgh 1987, pp 258-263

[24] H.-J. Wunderlich, "Multiple Distributions for Biased Random Test Patterns", Proc. International Test Conference ITC-88, Washington 1988, pp 236-244

[25] Y. Zorian, V. K. Agarwal, "Optimizing Error Masking in BIST by Output Data Modification", Journal of Electronic Testing: Theory and Application, Vol. 1, No. 1, Feb. 1990, pp 59-71