CONFIGURING FLIP-FLOPS TO BIST REGISTERS

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

Built-in self-test test registers must segment a circuit such that there exists a feasible test schedule. If a register transfer description is used for selecting the positions of test registers, the space for optimizations is small. In this paper, 1-bit test cells are inserted at gate level, and an initial test schedule is constructed. Based on the information of this schedule, test cells that can be controlled in the same way are assembled to test registers. Finally, a test schedule at RT level is constructed and a minimal set of test control signals is determined. The presented approach can reduce both BIST hardware overhead and test application time. It is applicable to control units and circuits produced by control oriented synthesis where an RT description is not available. Considerable gains can also be obtained if existing RT structures are reconfigured for self-testing in the described way.

KEYWORDS: Built-in self-test, register configuration, test registers, test scheduling

1. INTRODUCTION

1.1 Test registers for BIST

Built-in self-test is one of the most important techniques to test large and complex circuits. Test registers are added at the primary inputs and outputs of the circuit, and some additional test registers are inserted into the circuit. These multi-mode test registers generate patterns or compact test responses during test application (e.g. [13, 17, 26]).

In the test mode, the circuit is segmented into a set of subcircuits that are completely bounded by test registers (see figure 1). For testing 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 multiple input signature register (MISR), the block of logic connected to the inputs of this register, and a set of test registers to generate test patterns for the inputs of the block (cf. [8, 18]). If the collected signature differs from the correct signature, the circuit is faulty.

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 (response compaction), logic block 1, and the test registers T_1 and T_2 (pattern generation). The block contained in the test unit usually consists of combinational or pipeline structured logic. Test units may overlap.

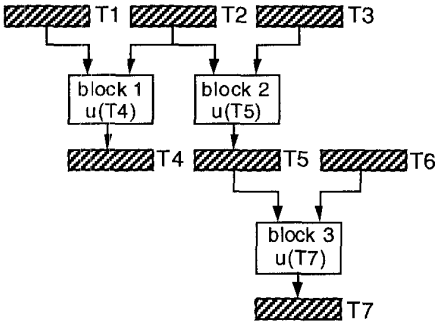

Figure 1: Example of test units at RT level (with test registers T_1, ..., T_7)

In order to obtain testable subcircuits, the test registers must be placed at appropriate positions. It has been shown independently by several authors that breaking all cycles in the circuit structure bounds the length of the required test sequences to the sequential depth of the circuit [4, 6, 11, 19, 20]. To keep the hardware overhead low the number of flip-flops that are integrated into test registers in order to break all cycles should be as small as possible. If the topology of the storage elements is represented by a so-called *S-graph* whose vertices correspond to flip-flops and whose edges indicate combinational paths between flip-flops, then this problem is equivalent to finding a "Minimum Feedback Vertex Set" [9]. Some authors also address extensions of this basic approach, as for example targeting a pipeline structure, limiting the sequential depth of the circuit, or considering timing constraints [4, 6, 11, 14, 19, 20].

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

0-7803-2102-2/94 $4.00

At RT level, the *register graph* $G_R := (V_R, E_R)$ is the counterpart of the S-graph. The nodes V_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 (see figure 2). The node set $V_R = V_N \cup V_T$ includes the registers V_N without testability features and the test registers V_T, which can generate test patterns and compact test responses.

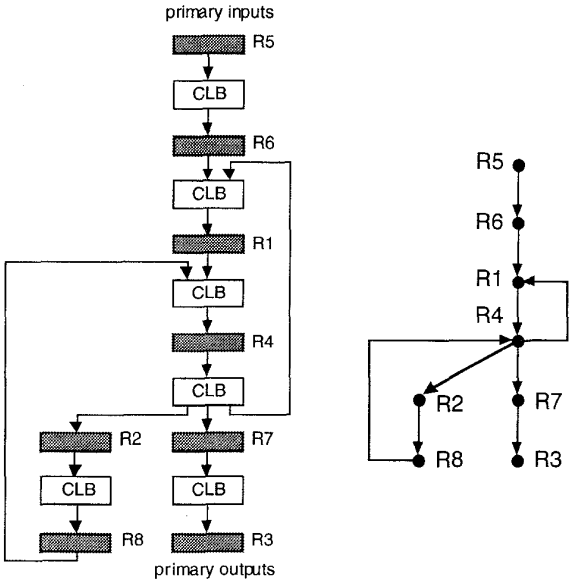

Figure 2: RT structure and corresponding register graph (CLB: combinational logic block, R•: register)

The *test register graph* $G_T := (V_T, E_T)$ is an abstraction of the register graph and describes the dataflow between the test registers. For each path in G_R that connects two nodes of V_T only via nodes of V_N there is a corresponding edge in E_T. If for instance $V_T = \{R_3, R_4, R_5\}$ (figure 2), then the test register graph is as shown in figure 3 and each cycle of the register graph contains one test register.

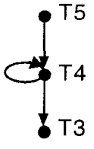

Figure 3: Test register graph (all cycles broken)

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 only feasible in some cases [16, 25], but in general the required properties of the test patterns cannot be guaranteed. In most cases the signatures are not exhaustive, (weighted) random or even deterministic, and an additional test register is required such that all cycles are broken at least twice. Hence for BIST the set V_T must include R_1, ..., R_5, and we obtain the test register graph of figure 4. The corresponding test units are shown in figure 5.

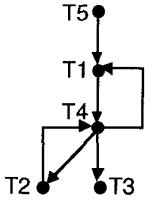

Figure 4: Test register graph for BIST

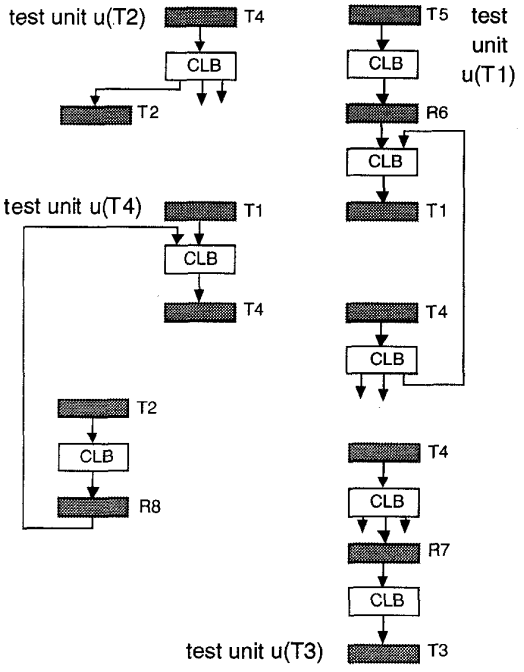

Figure 5: Test units for BIST

1.2 Test schedule

For test application, the order of testing all the test units must be determined. Generally not all test units can be tested simultaneously as they share some test resources that can be used only exclusively. These restrictions are described in the *test incompatibility graph* $G_I := (V_I, E_I)$ [8]. The nodes V_I of this graph represent the test units, the edges connect pairs of test units that cannot be tested simultaneously (incompatible test units).

The test incompatibility graph depends on the test strategy applied to the circuit under test. For example, pseudo-random testing requires that the output patterns of a test

register that performs signature analysis are not used as test patterns. Consequently, there is an edge $((u(T_i), u(T_j)) \in E_I$ if and only if the test register T_i is an input register (pattern generator) of $u(T_j)$ or the test register T_j is an input register of $u(T_i)$.

The test schedule can be structured in the following way. A *test session* s_i is a set of test units that are processed simultaneously. A *test schedule* $S := (s^r, O)$ is described by a sequence of test sessions $s := (s_0, s_1, ..., s_{d-1})$, a repetition number r, and a subset $O \subset V_T$ of test registers whose contents (signatures) are evaluated at test end. The set O 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 get a short test application time, a test schedule $S = ((s_0, s_1, ..., s_{d-1}), O)$ has to be determined where d is minimum. The set of test units must be partitioned into a minimum number of test sessions. This problem is equivalent to coloring the nodes of the test incompatibility graph with a minimum number of colors such that no edge connects two nodes of the same color [5, 8, 18]. 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.

1.3 Configuring flip-flops to test registers

A gate level circuit corresponds to a variety of different test register graphs. The test register graph is uniquely determined by the way the flip-flops are partitioned and assembled to test registers. Up to now, configuring flip-flops to test registers has not been intensively studied in literature. A top down design style has been assumed where the register graph is available as an intermediate structure. Then some of the registers have been transformed to test registers [1, 18, 21]. If there is a self-loop in the register graph, it is not possible to break all cycles twice and an additional test register must be included that is transparent in normal mode. These additional test registers may cause considerable hardware overhead.

As new design styles and synthesis procedures are applied, the top-down approach no longer leads to optimal results or is not even possible:

Control units: Control units form an increasing part of the circuits. Here the S-graph is strongly meshed, and an intermediate register transfer structure is not available. Examples are many of the ISCAS'89 benchmark circuits [2].

Control oriented synthesis: Some high level synthesis systems do not divide the system into data path and control unit (for an overview see [3]). As a consequence, the system contains both registers and single flip-flops that still need configuration to test registers.

General register transfer structures: The register configuration of the system mode is not always optimal for testing. As an example, figure 6 shows a carry save adder (CSA) and its register graph. Such a circuit is often used for implementing sequential multiplication [10].

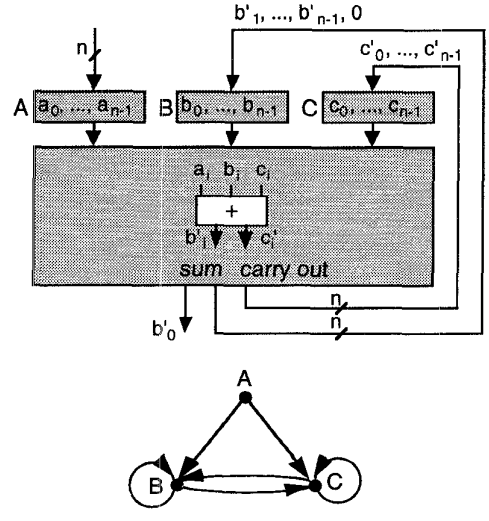

Figure 6: CSA data path and its register graph

The register graph contains two self-loops, and two additional transparent test registers B' and C' of length n are required for making it self-testable. Figure 7 shows the test register graph and the corresponding test incompatibility graph for random testing. The test schedule needs two test sessions.

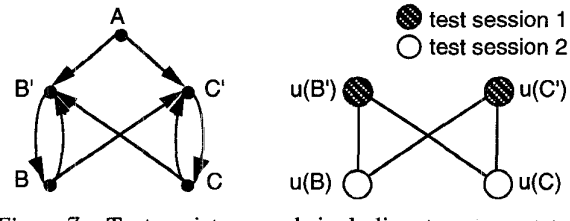

Figure 7: Test register graph including transparent test registers B' and C' (left), colored test incompatibility graph (right)

But looking at this circuit in more detail it is found that the transparent test register B' is superfluous. Flip-flop b_i just feeds flip-flops b_{i-1} and c_i, and flip-flop c_i feeds b_{i-1} and c_i; so the S-graph contains self-loops for the flip-flops c_i, but not for the flip-flops b_i. Hence it is more appropriate to split register B into two registers of length $\frac{n}{2}$

during testing, namely $B_0 := (b_0, b_2, ..., b_{n-2})$ and $B_1 := (b_1, b_3, ..., b_{n-1})$. The resulting register graph contains only one self-loop, and only one additional test register C' is required. The test incompatibility graph needs four colors (figure 8).

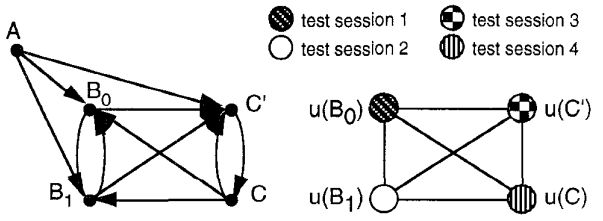

Figure 8: Test register graph after inserting test register C' and splitting register B (left), corresponding test incompatibility graph (right)

Up to now the hardware savings are paid by a longer test time. But if test register C is also split into $C_0 := (c_0, c_2, ..., c_{n-2})$ and $C_1 := (c_1, c_3, ..., c_{n-1})$, and the same is done with the transparent test register C', then two test sessions are sufficient (see figure 9).

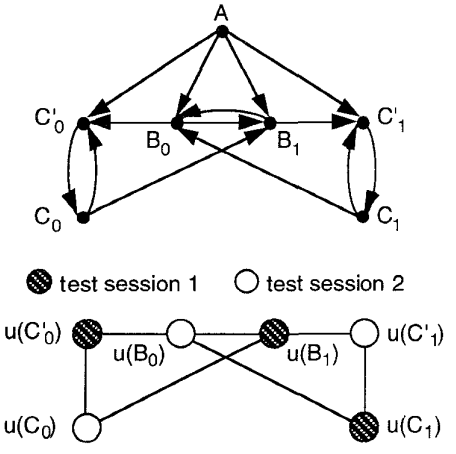

Figure 9: Test register graph after register reconfiguration, and corresponding test incompatibility graph

In this example, the hardware savings are obtained by cycle breaking at gate level and using this information for test register configuration. Also the test time is reduced by test scheduling at gate level before test register configuration.

1.4 Objectives of optimal test register configuration

Test register configuration is subject to a variety of objectives. First of all, the number of additional transparent test registers should be minimized. The other objectives are strongly related to BIST scheduling techniques and aim at minimizing the test time and reducing the hardware overhead. An optimal test register configuration should support test scheduling in the following ways:

- *Minimizing the number of test sessions:* Generally, a smaller number of test sessions reduces the overall test time. In addition, less area is required by the BIST control unit (e.g. [12]).
- *Minimizing the number of different signals for controlling the test registers:* Multi-mode test registers require at least two control signals: A signal TEST, which distinguishes between normal mode and test mode, and a signal **c**, which distinguishes between pattern generation and signature analysis in the test mode. All test registers may share the same TEST signal, but in general several signals **c** are required. The test control unit must generate these control signals for all the test registers. If the total number of different control signals is smaller, the test control unit can be implemented with smaller hardware cost. The area required for routing the control signals is reduced, too (e.g. [15]).
- *Reducing the number of signatures to be evaluated after testing:* The test registers are initialized only once at the beginning of the test. A test register that compacts test responses can get a faulty signature if the processed test unit contains a detectable fault, or if at least one of the involved pattern generating test 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 the signatures can influence one another, and the effects of a fault, namely faulty signatures, can propagate through the circuit [23]. This can be utilized to reduce the amount of self-test hardware, as for many circuits scanning and evaluation of signatures can be restricted to a subset of the test registers provided that the test schedule is constructed appropriately. Moreover, it is sufficient to scan the signatures only at the end of the test since any difference between the actual contents of a test register and the contents corresponding to the fault-free case will remain unchanged in the pattern generation mode.

The rest of the paper is organized as follows: In section 2, the circuit is made self-testable by placing test cells in the S-graph, and a test schedule is constructed at gate level. The information of this schedule is then exploited to determine maximal sets of test cells that can be controlled in the same way, and test registers are assembled from these sets (section 3). Another result of this step is a minimal set of control signals. In section 4, the schedule obtained at gate level is translated to a preliminary schedule at RT level, which is optimized such that the number of signatures to be evaluated at test end is

minimal. In section 5, all things are put together and the complete procedure is described, section 6 gives experimental results.

2. TEST SCHEDULING AT GATE LEVEL

At the gate level, the counterparts of the register graph G_R and test register graph G_T are the S-graph G_{R1} and the test cell or 1-bit test register graph G_{T1}. In order to have at least two test registers in each cycle of the register graph G_R, it is necessary to have at least two test register cells in each cycle of the underlying S-graph G_{R1}. Selecting the nodes of G_{R1} where test cells have to be inserted is similar to the problem of selecting the flip-flops of a partial scan path such that all cycles of the S-graph are broken twice. Therefore a sligthly modified version of the partial scan algorithms proposed in [11, 19] can be used. First an additional node is inserted into each self-loop of G_{R1}, then all the elementary cycles of G_{R1} are determined. A subset of nodes is chosen that contains at least two nodes of each cycle and is as small as possible. The nodes of this subset become test cells. In addition, test cells are inserted at all the primary inputs and outputs.

Figure 10 shows a simple example, which will be used throughout the paper to explain the proposed approach. Since the S-graph contains two self-loops, two storage elements that are transparent in the normal mode, r_{10} and r_{11}, have to be inserted. The storage elements r_3, r_4, r_6, r_{10}, and r_{11} are selected to become test register cells (e.g. 1-bit elements of a BILBO or a cellular automaton). Then each cycle of the S-graph contains two test cells. Further test cells are added at the primary inputs (r_1, r_2) and outputs (r_8, r_9).

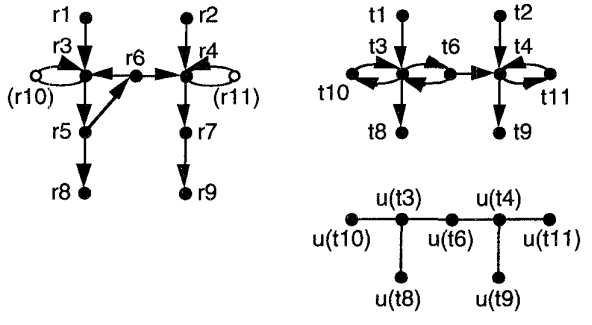

Figure 10: Example: S-graph G_{R1} (top left), test register graph G_{T1} (top right), test incompatibility graph G_{I1} assuming a pseudo-random test strategy (bottom)

The notion of test units can also be transferred to the gate level. Every "1-bit test unit" is defined by the test cell at its output. Moreover, similar to the test incompatibility graph G_I at RT level, a test incompatibility graph G_{I1} at gate level can be established, which represents the 1-bit test units and the pairs of 1-bit test units that must not be tested simultaneously.

Using these concepts, a test schedule based on 1-bit test units is constructed by graph coloring. For the example, one gets two test sessions, $s_0^{(1)} = \{u(t_3), u(t_4)\}$ and $s_1^{(1)} = \{u(t_6), u(t_8), u(t_9), u(t_{10}), u(t_{11})\}$.

During a test session, a test cell operates in the pattern generation mode or in the response compaction mode, or it is not used for testing. Thus, for a given sequence of test sessions, the operation of a test cell t has to be controlled according to a specific mode vector

$\mathbf{m_t} := (m_t(0), m_t(1), ..., m_t(d-1))$ where

$$m_t(j) := \begin{cases} 0 & \text{if test cell t generates patterns in session j} \\ 1 & \text{if test cell t compacts test responses in session j} \\ 2 & \text{else (test cell t not used in session j)} \end{cases}$$

for j = 0, 1, ..., d-1

The mode vector of a test cell at a primary input contains at least one component of value 0 and possibly some components of value 2. The mode vector of a test cell at a primary output consists of d-1 components of value 2 and one component of value 1 since the 1-bit test unit corresponding to this test cell is tested in exactly one test session. Finally, the mode vector of every other test cell contains at least one component of value 0, exactly one component of value 1, and possibly some components of value 2.

The mode vectors of the example are $\mathbf{m_{t_1}} = \mathbf{m_{t_2}} = (0, 2)$, $\mathbf{m_{t_3}} = \mathbf{m_{t_4}} = (1, 0)$, $\mathbf{m_{t_6}} = (0, 1)$, $\mathbf{m_{t_8}} = \mathbf{m_{t_9}} = (2, 1)$, $\mathbf{m_{t_{10}}} = \mathbf{m_{t_{11}}} = (0, 1)$.

3. SYNTHESIS OF TEST REGISTERS

In order to simplify test control, the operation of as many test cells as possible should be controlled by the same signals **c** that distinguish between pattern generation and signature analysis (see section 1.4). Test cells that are controlled by the same signal **c** can be included in the same test register.

The test cells at the primary inputs never have to compact test responses. Thus they can always operate in pattern generation mode, corresponding to the constant control signal **c** = (0, 0, ..., 0). Similarly, all the test cells at the primary outputs can be controlled by the constant signal **c** = (1, 1, ..., 1). These cells are assembled to separate test registers. The constant control signal makes it possible to reduce the hardware costs of these test registers.

In the following, only the remaining test cells are considered. Two test cells t_a and t_b can be controlled by the same signal if their mode vectors $\mathbf{m_{t_a}} := (m_{t_a}(0), ..., m_{t_a}(d-1))$ and $\mathbf{m_{t_b}} := (m_{t_b}(0), ..., m_{t_b}(d-1))$ satisfy the condition

$$\forall j \in \{0, 1, ..., d-1\} \; [m_{t_a}(j) + m_{t_b}(j) \neq 1]$$

i.e. there is no test session where one of the test cells must operate in pattern generation mode and the other in response compaction mode. Then the mode vectors $\mathbf{m_{t_a}}$ and $\mathbf{m_{t_b}}$ are said to be compatible.

A minimal number of different control signals is required if the mode vectors of the test cells are partitioned into maximal subsets of mutually compatible mode vectors and for each subset one control signal **c** is determined. Hence a graph G_M is constructed whose nodes represent the mode vectors and whose edges describe pairs of incompatible mode vectors. A minimal coloring of G_M gives the desired partition $\{\mu_0, \mu_1, ..., \mu_{k-1}\}$ of the mode vectors, where all mode vectors of a subset μ_i are mutually compatible. Since each mode vector has exactly one component of value 1 and all mode vectors having the value 1 in the same position are compatible, $k \leq d$ always holds.

Let $\mu_i = \{\mathbf{m_{t_1}}, ..., \mathbf{m_{t_n}}\}$ be one of these subsets. The values of the control signal $\mathbf{c}^{(\mu_i)}$ during the d sessions are computed by $\mathbf{c}^{(\mu_i)} := \mathbf{m_{t_1}} \oplus \mathbf{m_{t_2}} \oplus ... \oplus \mathbf{m_{t_n}}$ where the operator $\oplus$ is associative and defined by

$$(a_0, a_1, ..., a_{d-1}) \oplus (b_0, b_1, ..., b_{d-1}) = (c_0, c_1, ..., c_{d-1})$$

$$c_j := \begin{cases} a_j & \text{if } a_j = b_j \\ (a_j + b_j) \bmod 2 & \text{else} \end{cases}$$

for $j = 0, 1, ..., d-1$.

The partition $\{\mu_0, \mu_1, ..., \mu_{k-1}\}$ of the mode vectors induces a partition $\{\tau_0, \tau_1, ..., \tau_{k-1}\}$ of the test cells where every test cell $t \in \tau_i$ has a mode vector $m_t \in \mu_i$. Now from each subset τ_i we form one or more test registers that are controlled by $\mathbf{c}^{(\mu_i)}$.

The synthesized test registers should achieve low aliasing probabilities when they are used for test response compaction. Thus the size of the test registers must be sufficiently large, e.g. at least 16 or 20 bit. If a subset τ_i is smaller, sometimes the coloring of the graph G_M and thus the partition $\{\mu_0, \mu_1, ..., \mu_{k-1}\}$ can be modified such that the number of cells in τ_i is increased, or additional flip-flops must be included for test purposes. On the other side, if a subset τ_i is very large, several test registers can be built. Considering the hardware costs of configuring test cells, it is usually advantageous to group all the cells of τ_i that correspond to the same normal register into the same test register.

For the example, the maximal sets of compatible mode vectors are $\mu_0 = \{\mathbf{m_{t_3}}, \mathbf{m_{t_4}}\}$, $\mu_1 = \{\mathbf{m_{t_6}}, \mathbf{m_{t_{10}}}, \mathbf{m_{t_{11}}}\}$, and the control signals are $\mathbf{c}^{(\mu_0)} = (1, 0)$, $\mathbf{c}^{(\mu_1)} = (0, 1)$. We get the test registers $T_2 = \tau_0 = \{t_3, t_4\}$, $T_3 = \tau_1 = \{t_6, t_{10}, t_{11}\}$, $T_1 = \{t_1, t_2\}$ at the primary inputs, and $T_4 = \{t_8, t_9\}$ at the primary outputs. Figure 11 shows how the test control unit and the test registers are connected.

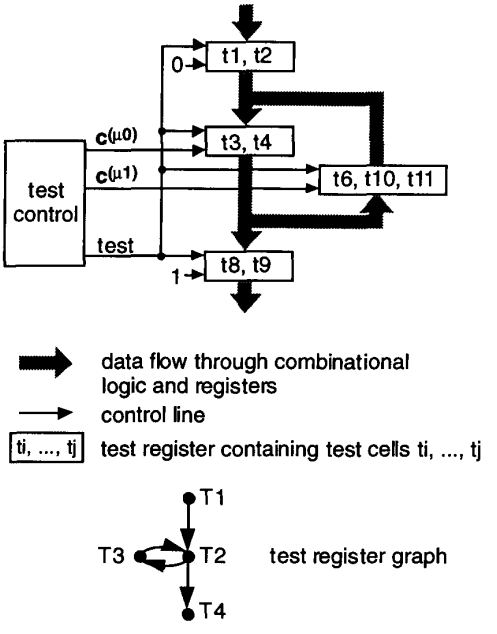

Figure 11: Synthesized test registers and test control unit for the example of figure 10

4. TEST SCHEDULING AT RT LEVEL

After test registers have been synthesized, the test sessions $s_0^{(1)}, s_1^{(1)}, ..., s_{d-1}^{(1)}$ constructed using 1-bit test units are translated to test sessions $s_0, s_1, ..., s_{d-1}$ that are based on the test units at RT level. For each 1-bit test unit $u(t_i)$ tested in session $s_j^{(1)}$, the test unit $u(T_h)$ with $t_i \in T_h$ has to be included in the session s_j. A test unit at RT level may be included in more than one test session. This is a consequence of exploiting don't cares when the number of control signals is minimized. The test sessions $s_0, s_1, ..., s_{d-1}$ and the control signals $\mathbf{c}^{(\mu_0)}, \mathbf{c}^{(\mu_1)}, ..., \mathbf{c}^{(\mu_{d-1})}$ together with the test lengths of the test sessions and the set of signatures to be evaluated give a behavioral specification of the test control unit. For the example, we get the test sessions $s_0 = \{u(T_2)\}$, $s_1 = \{u(T_3), u(T_4)\}$. A complete schedule is
$(\{\{u(T_2)\}, \{u(T_3), u(T_4)\}\}, \{T_2, T_3, T_4\})$.

Up to this point, we aimed at a minimal number of test sessions. In the following, the BIST hardware overhead is reduced by minimizing the number of signatures that have to be scanned and evaluated. The mutual influence of signatures is utilized in order to drive the test response information to the test registers at the primary outputs, and these can be accessed very easily.

Of course the signatures collected in the test registers at the primary outputs must always be evaluated since they cannot influence any other signatures. Only if some test registers at the primary outputs have a relatively high aliasing probability, other test registers should be added to the minimal set of test registers, O_{min}, whose contents have to be evaluated at test end. The test session sequence $(s_0, s_1, ..., s_{d-1})^r$ must be composed such that with repetition number r as small as possible the faults of all test units can influence the signatures in the test registers of O_{min}.

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 path imposes a condition on the test session sequence. For each test register that is used as a signature analyzer, *one path* to a test register of O_{min} needs to be considered. If there are multiple paths from a test register to test registers of O_{min}, a *shortest path* is selected.

The resulting conditions are 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 $u(T_j)$. A dummy node "end" is added to indicate the end of the test, and edges are inserted from all the nodes $u(T_h)$, $T_h \in O_{min}$, to the node "end". Figure 12 shows the test register graph, the precedence tree, and the test incompatibility graph for the example of figure 10.

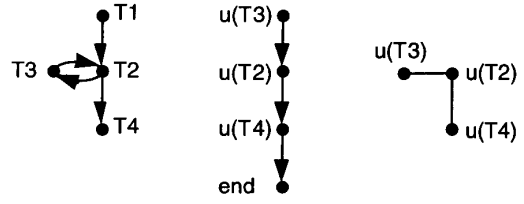

Figure 12: Example: Test register graph G_T (left), precedence tree P (center), and test incompatibility graph G_I (right)

The procedure CONSTRUCT_SEQ implements this scheduling approach (for a detailed description cf. [22]). As inputs it takes the precedence tree P, the test incompatibility graph G_I, and the period d of the test session sequence to construct. The results are the test session sequence s of length $d' \le d$ and the number r of repetitions necessary to propagate the effects of all faults to the test registers of O_{min}. For the example circuit described in figure 10 and figure 11, the resulting test schedules are $((\{u(T_2)\}, \{u(T_3), u(T_4)\})^2, \{T_4\})$ for d=2, and $((\{u(T_3)\}, \{u(T_2)\}, \{u(T_4)\}), \{T_4\})$ for d=3. If the test lengths of the test units are not extremely short and the test registers are sufficiently large (e.g. 20 bits or more), the probability of aliasing is very small and can be neglected even if faulty signatures are propagated through several test units [23].

In order to control the test registers according to the scheduling of CONSTRUCT_SEQ, a new set of control signals is required. The mode vectors of the test registers, maximal sets of compatible mode vectors and finally the control signals $c^{(\mu_0)}, c^{(\mu_1)}, ..., c^{(\mu_{d'-1})}$ can be determined in the same way as at gate level (see sections 2 and 3).

5. THE COMPLETE APPROACH

In this section, the ideas developed above are put together, and the complete approach is described.

Input:
- S-graph, $G_{R1} = (V_{R1}, E_{R1})$
- minimum size of test registers, n_{min}
- average size of test registers, n_{avg}

Output:
- Set of test registers, V_T
- test schedule, S

(1) for all self-loops of G_{R1}:
insert an additional (transparent) storage cell and modify G_{R1} accordingly;
/* result: $G'_{R1} = (V'_{R1}, E'_{R1})$ */

(2) determine a minimal subset of nodes $V_{T1} \subset V'_{R1}$ that contains all primary inputs and outputs and at least 2 nodes of each cycle of G'_{R1};
/* test register cells */

(3) construct the 1-bit test register graph $G_{T1} = (V_{T1}, E_{T1})$;

(4) build the test incompatibility graph G_{I1} based on G_{T1} and the chosen test strategy;

(5) GRAPHCOLOR (in: G_{I1}, out: $\{s_0^{(1)}, s_1^{(1)}, ..., s_{d-1}^{(1)}\}$);
/* test sessions based on "1-bit test units" */

(6) for each test register cell t:
determine the mode vector $\mathbf{m_t}$ according to the test session sequence ($s_0^{(1)}, s_1^{(1)}, ..., s_{d-1}^{(1)}$);

(7) build the mode incompatibility graph G_M;

(8) GRAPHCOLOR (in: G_M, out: $\{\mu_0, \mu_1, ..., \mu_{k-1}\}$);
/* maximal sets of compatible mode vectors */

(9) for each set μ_i:
compute the control signal $\mathbf{c}^{(\mu_i)}$;
/* control of test register cells */

(10) partition the set of test register cells into k subsets $\tau_0, \tau_1, ..., \tau_{k-1}$ such that the mode vectors of the test register cells of subset τ_i are the mode vectors of μ_i, i = 0, ..., k-1;

(11) for each set τ_i:
if $|\tau_i| < 2 \cdot n_{min}$
form one test register including all the cells of τ_i;
else
form $\left\lfloor \frac{|\tau_i|}{n_{avg}} \right\rfloor$ or $\left\lceil \frac{|\tau_i|}{n_{avg}} \right\rceil$ test registers that each contain about n_{avg} cells of τ_i;
/* result: set of test registers V_T */

(12a) translate the test sessions $s_0^{(1)}, s_1^{(1)}, ..., s_{d-1}^{(1)}$ to test sessions $s_0, s_1, ..., s_{d-1}$ at RT level;
S := $((s_0, s_1, ..., s_{d-1}), T)$; /* test schedule with minimal number of test sessions */

or alternatively

(12b) build the test register graph G_T, the precedence tree P, and the test incompatibility graph G_I;
CONSTRUCT_SEQ (in: P, G_I, d; out: **s**, d', r);
S := $((s_0, s_1, ..., s_{d-1})^r, O_{min})$;
/* test schedule where only the signatures of $O_{min} \subset T$ have to be evaluated */
compute new control signals $\mathbf{c}^{(\mu_0)}, \mathbf{c}^{(\mu_1)}, ..., \mathbf{c}^{(\mu_{d'-1})}$. /* control of test registers */

The graph coloring problems that have to be solved in step (5) and step (8) are NP-complete problems [9]. But many efficient heuristics are known that give good (suboptimal) solutions. We applied the algorithm of [7, pp. 70-71]. This algorithm first determines an initial coloring using a greedy strategy and then tries to improve this solution by recoloring some nodes. All possible solutions are implicitly enumerated. Solutions with 1, 2, or 3 colors are guaranteed to be a minimum coloring.

6. EXPERIMENTAL RESULTS

The described procedures have been applied to the large circuits of the ISCAS'89 benchmark set [2]. For BIST, test cells were added at the primary inputs and outputs, and additional test cells were inserted such that each cycle of the circuit structure contained at least two of them. The test incompatibility graph for the 1-bit test units, G_{I1}, was constructed assuming pseudo-random testing. Then the nodes of this graph were colored using a minimal number of colors. In those cases where exhaustive search for a minimum coloring took too much time, we stopped the recoloring process after 10000 trials to improve the solution and used the best solution found so far.

The result of these first steps is a minimal number of test sessions that are based on 1-bit test units (see table 1). The number of different mode vectors and the minimized number of signals necessary to control the test cells are also given in table 1. (Here and in the following tables we count only the nonconstant mode vectors and control signals that are required for switching the test cells between pattern generation mode and response compaction mode.)

circuit	# test cells	# test sessions composed of "1-bit test units"	# mode vectors	# control signals
s9234	346	2	2	2
s13207	784	3	7	3
s15850	986	3	5	3
s35932	967	3	6	3
s38417	2317	3	6	3
s38584	2521	6	33	5

Table 1: Test scheduling at gate level and minimization of control signals

Subsequently, test registers were assembled by combining test cells that are controlled by the same signal (table 2).

circuit	# test registers (optimal)	width of test registers (bit)		
		minimal *	average	maximal
s9234	12	22	28.8	31
s13207	24	30	32.7	34
s15850	32	29	30.8	33
s35932	30	31	32.2	40
s38417	72	31	32.2	36
s38584	80	30	31.5	39

Table 2: Optimally synthesized test registers
(* except test registers at the primary inputs)

For comparison, we omitted test scheduling at gate level and combined randomly chosen test cells to test registers.

In this case only the compatibility conditions represented by the graph G_{I1} were taken into account. Again, the test cells at the primary inputs and outputs were assembled to separate test registers. Table 3 shows the statistics of the randomly assembled test registers. They have about the same sizes as the test registers of table 2.

circuit	# test registers (random)	width of test registers (bit)		
		minimal *	average	maximal
s9234	12	22	28.8	32
s13207	25	21	31.4	34
s15850	32	22	30.8	33
s35932	30	27	32.2	35
s38417	72	24	32.2	36
s38584	81	30	31.1	33

Table 3: Test registers formed randomly
(* except test registers at the primary inputs)

Based on the assembled test registers and the test units defined by them, test schedules can be constructed in two different ways. One objective is minimizing the number of test sessions, the other objective is a minimal number of signatures to be evaluated.

In order to get a minimal number of test sessions the well-known method of [8] can be applied to the circuit at RT level. With our approach, however, we have already got a test schedule based on 1-bit test units (see table 1) that optimally matches with the synthesized test registers. This gate level test schedule can be translated to a RT level test schedule with the same minimal number of test sessions and control signals. In table 4, the results of the presented algorithm (column "optimal") are compared with test schedules that were obtained by applying the method of [8] to the circuits with randomly assembled test registers (column "random").

circuit	# test sessions (d)		# control signals	
	random	optimal	random	optimal
s9234	11	2	10	2
s13207	21	3	20	3
s15850	27	3	26	3
s35932	20	3	19	3
s38417	64	3	64	3
s38584	67	6	66	5

Table 4: Test schedules with minimal number of test sessions

Test registers that are synthesized optimally for BIST give significant advantages both in terms of silicon area and test length. Built-in self-test requires additional hardware for

- test registers
- test control unit
- distribution of test control signals

The presented approach achieves benefits at all three of these points. It minimizes the number of test cells and needs at most as many test cells as the approaches that insert BILBO-like test registers at RT level. It minimizes the number of test control signals and the number of test sessions. This simplifies test control since the control unit must generate the specified values of the control signals for all test sessions. The small number of control signals also reduces the area overhead for distributing these signals to the test registers.

Moreover, reducing the number of test sessions usually leads to a shorter overall test length. Of course, the test length also depends on the type of the test registers and on the fault coverage value that has to be achieved. The method presented is compatible with different kinds of test registers [24]. For example, using test registers that can produce weighted random patterns generally results in a shorter test length for the considered test unit than using unbiased random patterns.

As an alternative to test schedules with minimal number of test sessions, the procedure CONSTRUCT_SEQ gives schedules where all the test response information is driven to the primary outputs, and as a consequence only the few signatures at the primary outputs have to be evaluated (see table 5).

circuit	# executed test sessions (r*d)		# control signals		# evaluated signatures	
	random	opt.	rand.	opt.	rand.	opt.
s9234	1 * 11	2 * 2	10	2	1	1
s13207	1 * 21	1 * 4	20	3	4	4
s15850	1 * 27	1 * 4	26	3	3	3
s35932	1 * 20	1 * 4	19	3	10	10
s38417	1 * 65	1 * 6	64	5	3	3
s38584	1 * 67	1 * 7	66	6	9	9

Table 5: Test schedules constructed by CONSTRUCT_SEQ

Compared to the schedules with minimal number of test sessions (cf. table 4, r=1), the number of executed test sessions (r*d) is larger. But further hardware savings can be achieved since the number of signatures that have to be evaluated is cut down. As all the test registers used as signature analyzers have at least 21 bits, the probability of aliasing can be neglected and fault coverage is (practically) not affected by reducing the number of evaluated signatures. Moreover, we see that with the randomly assembled test registers the number of different test sessions (d) and the number of executed test sessions (r*d) grow by a factor

of 3 to 11. The required number of control signals is 5 to 13 times larger. This again underlines the advantages of test registers that are optimally synthesized for BIST.

7. CONCLUSIONS

Usually test register insertion and test scheduling are considered separately. In this paper, both problems are tackled together in order to find a global optimum. A method has been presented that groups flip-flops of the original design to test registers. The number of additional test cells that are needed to break self-loops is minimized. The proposed test register configuration minimizes the number of different test sessions and the number of control signals required to control the test registers. The effects are reduced area of the test control unit, reduced area for routing the control lines, and shorter test application time. The final test scheduling process at RT level can cut down the number of signatures that have to be evaluated at test end.

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

[3] R. Camposano, R. A. Walker: "A Survey of High-Level Synthesis Systems", Kluwer Academic Publishers, Norwell MA, 1991

[4] K.-T. Cheng, V. D. Agrawal: "A Partial Scan Method for Sequential Circuits with Feedback", IEEE Trans. on Computers, Vol. 39, No. 4, April 1990, pp. 544- 547

[5] C.-I. H. Chen: "Graph Partitioning for Concurrent Test Scheduling in VLSI Circuit", Design Automation Conference, San Francisco, 1991, pp. 287-290

[6] V. Chickermane, J. H. Patel: "An Optimization Based Approach to the Partial Scan Problem", Int. Test Conference, Washington D.C., 1990, pp. 377-386

[7] N. Christofides: "Graph Theory - An Algorithmic Approach", Academic Press, London, 1975

[8] 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, Sept. 1988, pp. 1099-1109

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

[10] D. Goldberg: "Computer Arithmetic", in: J. L. Hennessy, D. A. Patterson: "Computer Architecture: A Quantitative Approach", Morgan Kaufmann, San Mateo CA, 1990

[11] R. Gupta, R. Gupta, M. A. Breuer: "The BALLAST Methodology for Structured Partial Scan Design", IEEE Transactions on Computers, Vol. 39, No. 4, April 1990, pp. 538-544

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

[13] P. D. Hortensius et al.: "Cellular Automata-Based Pseudorandom Number Generators for Built-In Self-Test", IEEE Trans. on CAD, 1989, pp. 842-859

[14] J.-Y. Jou, K.-T. Cheng: "Timing Driven Partial Scan", Int. Conference on Computer-Aided Design, Santa Clara CA, 1991, pp. 404-407

[15] J. Kalinowski, A. Albicki, J. Beausang: "Test Control Signal Distribution in Self-Testing VLSI Circuits", Int. Conf. on CAD, 1986, pp. 60-63

[16] 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.

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

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

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

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

[21] S.-P. Lin, C. A. Njinda, M. A. Breuer: "A Systematic Approach for Designing Testable VLSI Circuits", Int. Conference on CAD, Santa Clara CA, 1991, pp. 496-499

[22] A. P. Stroele, "Self-Test Scheduling With Bounded Test Execution Time", Int. Test Conference, Baltimore MD, 1992, pp. 130-139

[23] A. P. Stroele, H.-J. Wunderlich: "Error Masking in Self-Testable Circuits", Int. Test Conference, Washington D.C., 1990, pp. 544-552

[24] A. P. Stroele, H.-J. Wunderlich: "A Unified Method for Assembling Global Test Schedules", Third Asian Test Symposium, Nara, Japan, 1994

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

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