Correspondence

The BALLAST Methodology for Structured Partial Scan Design

RAJESH GUPTA, RAJIV GUPTA, AND MELVIN A. BREUER

Abstract—**A new efficient partial scan technique is presented. Scan path storage elements (SPSE's) are selected such that the remainder of the circuit has certain desirable testability properties. A complete test set is obtained using combinational ATPG. Some SPSE's may need to be provided with a HOLD mode; their number is minimized by ordering the registers in the scan path and formatting the test patterns appropriately. This methodology leads to a low area overhead and allows 100% coverage of irredundant faults.**

Index Terms—**Balanced sequential structures, design for testability, partial/incomplete scan design, sequential circuit testing, single-pattern testability, test pattern formatting.**

I. Introduction

Automatic test pattern generation (ATPG) for sequential circuits is generally considered to be a hard problem. Full scan design techniques attempt to alleviate this problem by connecting all flip-flops (FF's) or latches into a scan path during test mode so that all these elements become easily controllable and observable. Thus, in a circuit designed using full scan the portion of the circuit excluding the scan path, which we shall refer to as the *kernel* of the circuit, is fully combinational. Due to the overhead in modifying the FF's, partial scan techniques [1] have been proposed in which only a subset of the FF's are included in the scan path. This implies that the kernel is itself sequential. Hence, these techniques either require sequential ATPG [2], [3] or use combinational ATPG techniques without covering all faults in the kernel [4].

In this paper, we present an alternative partial scan methodology, BALLAST (BALAnced structure Scan Test), that requires only combinational ATPG and attains complete coverage of all detectable faults. Scan path FF's are selected in such a way that the resulting kernel belongs to a certain class of easily testable structures called *B-structures*. Test patterns for the kernel are obtained by treating it as being combinational, with FF's within it simply replaced by delayless wires. During test application, each pattern is *held constant* in the scan path for a fixed number of clock cycles before the output pattern of the kernel is sampled and shifted out.

The BALLAST methodology is illustrated using the circuit of Fig. 1 consisting of six registers and four combinational logic blocks. The only registers in the scan path are $R3$ and $R6$. In this example, the kernel has a pipelined structured. For the time being, let us assume that the scan path registers should have the ability to hold data constant across consecutive clock cycles (later this requirement will be relaxed). Test patterns can be obtained by using combinational ATPG in the manner described above. In order to test the kernel, each test pattern is shifted into the scan path and held in it for two clock cycles before the kernel output is loaded back into the scan path and shifted out. We will prove that using this approach we can detect all faults in the kernel that can cause errors in the logical operation of the circuit. We shall also study the implications of "unbalanced" paths in the kernel with unequal delays, and of the presence of HOLD modes in nonscan registers for normal system operation.

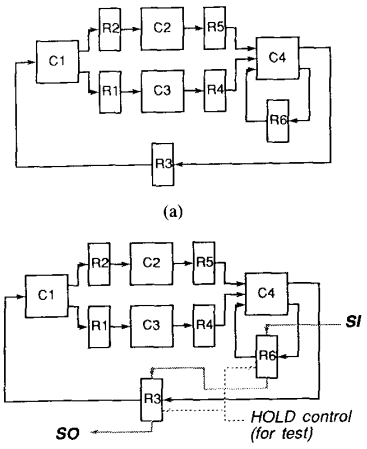

Fig. 1. Illustration of BALLAST methodology. (a) Synchronous circuit. (b) Partial scan design.

A partial scan technique with some similarities to BALLAST was developed independently by Kunzmann [5]. This technique has several limitations which are addressed by BALLAST. First, the nonscan storage elements need to be made transparent in test mode, which diminishes the advantage of using partial scan. Second, nonscan storage elements are passive during test (since they are bypassed) and do not get exercised. Third, the technique is not applicable if any storage elements have LOAD ENABLE control signals.

II. *B*-Structures and their Properties

In this paper, we consider only synchronous sequential circuits in which every cyclic path contains at least one clocked storage element. The storage elements are assumed to be edge-triggered flip-flops (FF's). The circuit may have any number of clocks. However, the FF clock signals must all be controlled by primary inputs, and no clock signal may feed the data input of any FF either directly or through combinational logic. The FF's may have LOAD ENABLE control signals, with similar restrictions as for the clock signals. The FF's may also have RESET/PRESET controls provided they are present in all FF's and all are controlled by the same primary RESET/PRESET control signal.

Circuit Model

In general, a synchronous sequential circuit S consists of blocks of combinational logic connected with each other either directly or through registers. A *register* is defined as a collection of one or more FF's with all FF's driven by the same clock signal and controlled by the same mode control signal (if any). Any subset of the FF's in a register forms a valid register.

The set of registers in the circuit can be partitioned into two sub-

Manuscript received July 6, 1989; revised November 27, 1989. This work was supported in part by the Defense Advanced Research Projects Agency and monitored by the Office of Naval Research under Contract N00014-87-K-0861, and in part by the Semiconductor Research Corporation under Contract 88-DP-075. The views and conclusions contained in this paper are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the U. S. Government.

The authors are with the Department of Electrical Engineering-Systems, University of Southern California, Los Angeles, CA 90089.

IEEE Log Number 8933872.

0018-9340/90/0400-0538$01.00 © 1990 IEEE

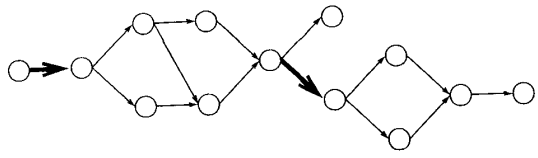

Fig. 2. Example of topology graph.

sets based on the presence or absence of explicit LOAD ENABLE controls on the FF's comprising them. We define the *load set* L as the set of registers in the circuit whose FF's have no explicit load enable control; these registers always operate in the LOAD mode (in which data are read from the data input during every clock cycle). Similarly, we define the *hold set* H as the set of registers whose FF's have an explicit LOAD ENABLE control signal; these registers have two modes of operation: a HOLD mode (in which they retain their value across consecutive clock cycles) as well as a LOAD mode.

The combinational logic in S can be partitioned into *clouds*, where each cloud is a maximal region of connected combinational logic such that its inputs are either primary inputs or outputs of FF's and its outputs are either primary outputs or inputs to FF's. In Fig. 1(a) each block of combinational logic $C1$, $C2$, $C3$, $C4$ represents a cloud.

From the way in which clouds are defined, it follows that no two clouds can be directly connected together; they must be separated by one or more registers. Furthermore, each FF in a register must receive data from exactly one cloud and must feed exactly one cloud. We can, therefore, constrain the grouping of FF's into registers (by splitting registers where necessary) such that each register receives data from exactly one cloud and feeds exactly one cloud. Under this constraint, the topology of the circuit can be modeled by a directed *topology graph* $G = (V, A, H, w)$ in which nodes in V represent clouds, each arc in A represents a connection between two clouds through a register, arcs in $H \subset A$ represent HOLD registers, and w: $A \rightarrow \mathbb{Z}^+$ (positive integers) defines the number of FF's in each register. We will use $w(a)$ to represent the cost of converting the register a into a scan path register. Fig. 2 shows a typical topology graph; bold arcs represent HOLD registers and others represent LOAD registers.

B-Structures

Let S be an arbitrary synchronous sequential circuit with topology graph $G = (V, A, H, w)$.

Definition: S is said to be a *balanced sequential structure* (*B-structure*) if

1) G is acyclic;

2) $\forall v_1, v_2 \in V$, all directed paths (if any) from v_1 to v_2 are of equal length (this actually includes condition 1); and

3) $\forall h \in H$, if h is removed from G, the resulting graph is disconnected. □

The example of Fig. 2 satisfies these conditions and is therefore a B-structure.

Given a balanced sequential structure S^B, we define its *combinational equivalent* C^B is the combinational circuit formed by replacing each FF in every register in S^B by a wire (if the output of the register uses the Q output of the FF) or an inverter (if it uses the $\bar{Q}$ output of the FF) or both. Define the *depth* d of S^B as the longest directed path in its topology graph. Given an input pattern I applied to S^B, define the *single-pattern output* of S^B for I as the steady-state output of S^B when I is held constant at the inputs to S^B and all its registers are operated in LOAD mode for at least d clock cycles. Given some fault f in S^B, if the single-pattern outputs for I of the good and the faulty circuits are different, then I is a *single-pattern test* for f.

B-structures have two interesting properties which allow them to be used as kernels in a BALLAST partial scan design: 1) they are single-pattern testable, and 2) a complete single-pattern test set can be derived using combinational test generation techniques. Both properties will be proved in Section IV. Next we present an overview of the proposed BALLAST methodology.

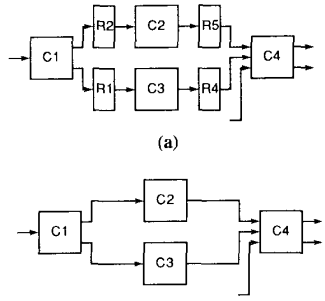

Fig. 3. Kernel and combinational equivalent of Fig. 1(b). (a) Kernel. (b) Combinational equivalent.

III. Scan Design Using *B*-Structures

When any register in a sequential circuit is included in a scan path, it serves as a control and observation point for the rest of the circuit. In effect it becomes a primary output of the cloud feeding it and a primary input of the cloud it drives. Thus, in our circuit model, the inclusion of a register in the circuit scan path corresponds to its removal from the topology graph of the circuit; the reduced topology graph represents the kernel, i.e., the portion of the circuit to be tested using the scan path.

The following is an outline of the BALLAST methodology.

1) Construct the topology graph G of the circuit as defined in Section II.

2) Select a minimal cost set of arcs R to be removed from G such that the remaining topology graph is balanced. Arcs in R represent the registers that must be included in the scan path. Let S^B be the B-structure corresponding to the resulting topology graph, which represents the kernel of the circuit. For the example of Fig. 1 the kernel is the B-structure shown in Fig. 3(a). Algorithms for implementing this step are presented in Section V.

3) Determine the combinational equivalent C^B of the kernel S^B. The combinational equivalent of the kernel in Fig. 3(a) is shown in Fig. 3(b). Using traditional combinational ATPG, determine a complete test set T for C^B. Since S^B is balanced, T will constitute a complete single-pattern test set for all detectable faults in the kernel S^B. (See Section IV for proof of correctness of this statement.)

4) Construct a scan path by appropriately ordering the registers in R and connecting them so that they are capable of a) shifting test patterns in/out, b) holding a pattern constant at the kernel inputs for d clock cycles (where d is the depth of the B-structure comprising the kernel), and c) loading in the test results from the kernel. Requirement b) can be achieved by providing all scan registers with a HOLD mode as in Fig. 1(b). In Section VI, we will show how the need for all scan registers to have a HOLD mode can be partially removed.

Fig. 4 shows two illustrations of the above methodology which are variations on the example of Fig. 1. In Fig. 4(a), the register $R1$ has a HOLD mode provided for normal system operation. Fig. 4(c) shows the kernel when $R1$, $R3$, and $R6$ are made scannable; clearly it is a B-structure and hence a valid kernel. Fig. 4(b) shows a second variation in which the two paths between $C1$ and $C4$ have unequal delays. By scanning $R1$, $R3$, and $R6$, one of the paths is broken and the resulting kernel in Fig. 4(d) is again a B-structure.

Given a circuit designed in the manner described above, the test plan for applying a sequence of single-pattern tests to the circuit is as follows. N is the number of test patterns to be applied and l is the total number of FF's in the scan path.

1) Operate all scan registers in the SHIFT mode for l clock cycles. (Scan in the first test pattern.)

2) Repeat N times:

   a) Place all scan registers in the HOLD mode and all nonscan registers in the LOAD mode for d clock cycles. (This allows test data to propagate through the kernel.)

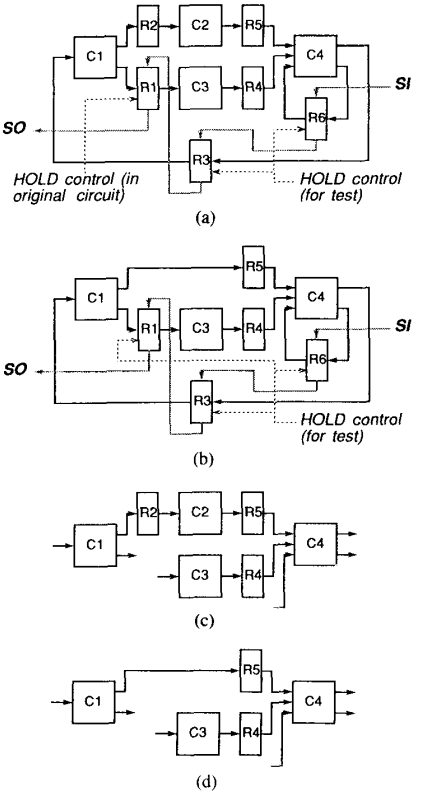

Fig. 4. Additional illustrations of the methodology. (a) Example with HOLD mode. (b) Example with unbalanced paths. (c) Kernel of (a). (d) Kernel of (c).

b) Operate all scan registers in the LOAD mode for 1 clock cycle. (Load the test result into the scan registers.)
c) Operate all scan registers in the SHIFT mode for l clock cycles. (Scan out the test result and scan in the next test pattern.) □

The partial scan technique described in this section is applicable to general sequential circuits. It is particularly well suited to pipelined circuits with limited feedback and feedforward connections. On the other hand, it is not very effective for circuits in which there are few clouds or in which every storage element is connected in a feedback loop through a single cloud; in the latter case, the technique reduces to full scan.

IV. Proof of Correctness

In this section, the testability properties of B-structures are studied with the objective of proving that any B-structure is a valid kernel in the BALLAST test methodology. We shall focus on the kernel of the circuit under consideration in isolation from the scan path. We shall treat all connections of the kernel from/to scan path registers as primary I/O lines of the kernel. The BALLAST methodology ensures that the kernel is a B-structure; this has the following implications. Despite the kernel being sequential, the fact that it is acyclic greatly simplifies the process of obtaining a test sequence [6, Section 3.5]. But this fact is of limited practical importance to us since arbitrary multiple-pattern test sequences cannot be efficiently applied to the kernel using a serial scan path. However, by restricting the kernel to being a B-structure, we shall demonstrate that single-pattern tests exist for all detectable faults in the kernel and that a complete test set of this type can be obtained using ordinary combinational ATPG on the combinational equivalent of the kernel.

Let S^B be an arbitrary B-structure and C^B its combinational equivalent. The set of faults of a B-structure refers to the union of the sets of single stuck-at faults of the individual clouds. A stuck-at fault in a register can be considered to be equivalent to a stuck-at fault in one of the clouds adjacent to it. A *detectable* fault is one that can be detected by some sequence of test patterns.

By definition of the combinational equivalent, S^B can be viewed as the combinational circuit C^B with delays introduced appropriately within it. Since delays do not affect the steady-state output function of combinational logic, we have the following lemma.

Lemma 1: For any input pattern I, the output of C^B and the single-pattern output of S^B are identical.

This lemma holds for all acyclic structures; a formal proof by induction on the depth of S^B can be found in [7].

Lemma 2: Let f_S be a fault in S^B and let f_C be the corresponding fault in C^B. Then any test pattern t for f_C in C^B is a single-pattern test for f_S in S^B.

Proof: Let C^B_f and S^B_f be the faulty circuits produced by f_C and f_S, respectively. Since t detects f_C, the outputs of C^B and C^B_f must differ for input t. Due to Lemma 1, the single-pattern outputs of S^B and S^B_f must differ for input t. Thus, t is a single-pattern test for f_S in S^B. □

Note that the above lemma does not prove that there is a single-pattern test for *every* detectable fault in S^B.

Theorem 1: Every B-structure is fully testable for all *detectable* faults using single-pattern tests.

Proof: Let the B-structure S^B have depth d and let its combinational equivalent be C^B. Let f be a detectable single stuck-at fault in S^B and let T be a test for f. T is in general a sequence of patterns to be applied to all data as well as control inputs of S^B. It is sufficient to prove that T can be transformed into a single-pattern test for f.

Let T cause the fault to be first detected at a particular output z of S^B. Consider each data input x_i of S^B (i.e., any input other than a control input for a register). Since S^B is balanced, all paths from x_i to z must have equal delays. (Note that since S^B is a B-structure, if there is a HOLD register in any of these paths then all the paths must pass through it, thus preserving equal delays.) This implies that the output value at z when the fault is detected depends on the value at x_i during at most one clock cycle, say τ_{x_i}. Hence, if we transform T to T' such that for each input x_i, the value required at clock cycle τ_{x_i} in T is applied at every clock cycle in the sequence T', then T' must be a valid test for f. We can now transform T' into T'' by operating all HOLD registers in S^B in the LOAD mode throughout the test. Note that the final output values at z for T' and T'', respectively, must be identical, since both can be simulated by applying the corresponding input pattern to C^B. It follows that the resulting sequence T'' must also detect f by producing an erroneous output at z. Hence, any test sequence can be transformed into a single-pattern test. □

A detailed version of the above proof can be found in [8]. In the test T'' derived above, all registers operate in the LOAD mode; this leads to the following corollary.

Corollary: The maximum required duration of any single-pattern test is d clock cycles, where d is the depth of the B-structure. □

We now proceed to show that a complete single-pattern test set for a B-structure can be obtained using ordinary combinational ATPG. Lemma 2 showed that any fault that is detectable in C^B is single-pattern testable in S^B. It remains to be shown that *every* detectable fault in S^B is detectable in C^B.

Lemma 3: Let f_S be a fault in S^B and let f_C be the corresponding fault in C^B. If f_S is detectable in S^B then f_C is detectable in C^B.

Proof: Let C^B_f and S^B_f be the faulty circuits. Since f_S is detectable in S^B, by Theorem 1 there must be a single-pattern test t for this fault. Hence, S^B and S^B_f must have different single-pattern outputs for t. This implies, by Lemma 1, that C^B and C^B_f must have different outputs for t. Thus, t is a test for f_C in C^B. □

Theorem 2: Given a balanced structure S^B, any complete test set for all detectable faults in its combinational equivalent C^B is a complete single-pattern test set for all detectable faults in S^B.

Proof: Let T be a test set for all detectable faults in C^B. We need to show that T is a single-pattern test set for all detectable faults in S^B.

Let f_S be detectable fault in S^B and f_C be the corresponding fault in C^B. Since f_S is detectable, by Lemma 3 f_C must also be

detectable. Since T detects all detectable faults in C^B, it must contain some pattern t that detects f_C. Hence, by Lemma 2, t must detect f_S. Thus, T detects all detectable faults in S^B. □

The two theorems reduce the problem of test generation for B-structures to the simpler problem of combinational ATPG, from which a complete single-pattern test set can be obtained. The single-pattern nature of the test sequences makes it possible, in a partial scan circuit, to easily test a kernel that is a B-structure. Each single-pattern test can be shifted into the scan path and held constant for the required number of clock cycles before the test results are loaded into the scan path and shifted out. Note that there is no loss in single stuck-at fault coverage coverage when this form of partial scan is used in place of full scan. The implementation issues related to the scan path will be discussed in Section VI.

V. Algorithm for Scan Register Selection

Given an arbitrary sequential circuit to which partial scan is to be applied, BALLAST selects a set of registers to be made scannable such that the kernel is a B-structure and the cost of modifying the circuit is minimized. It assumes that the modification cost is the same for registers having equal width, except in the case of a tie between a LOAD register and a HOLD register. In this case, the latter is chosen to be in the scan path because converting a HOLD register into a scan register may require a lower area. The selection is carried out using the algorithms presented in this section.

Let $G = (V, A, H, w)$ be the topology graph of the circuit. Formally, we need to determine a set of registers $R \subseteq A$ such that the topology graph of the kernel, $(V, A - R, H - R, w)$, is balanced and $\Sigma_{a \in R}\, w(a)$ is minimized. The solution can be obtained using the following steps.

Step 1. Transform $G = (V, A, H, w)$ into an acyclic topology graph G_A by removing a set of "feedback" arcs R_A such that $\Sigma_{a \in R_A}\, w(a)$ is minimized.

Step 2. Transform G_A into a balanced topology graph G_B by removing a set of arcs R_B such that $\Sigma_{a \in R_B}\, w(a)$ is minimized.

Step 3. $R = R_A \cup R_B$ is the desired set of arcs, and the resulting topology graph $G_B = (V, A - R, H - R, w)$ represents the kernel.

Note that because the problem is partitioned in this way, optimal solutions for the individual steps do not imply an optimal overall solution.

The problem of carrying out Step 1 above is known to be NP-complete [9]. Our implementation of BALLLAST uses a branch-and-bound algorithm adapted from [10], to which the reader is referred for details.

Balancing Acyclic Sequential Structures

A simplified form of Step 2 above has also been shown to be computationally intractable [5]. In this section, we present a heuristic procedure, *balance*, for solving this problem. *balance* uses a verification procedure, *check*, to verify that a given structure is balanced.

Verification Procedure: The procedure *check* first verifies that the removal of each arc in the hold set would disconnect the topology graph. Then, starting at each root node (i.e., node with no incoming arcs) in turn, it levelizes the portion of the graph reachable from it. If every node is found to be at a unique distance from each root, the graph is pronounced balanced and the procedure returns SUCCESS. If an imbalance is detected at any time, the procedure exits and returns FAILURE. The detailed procedure is listed in [7]. Its computation complexity is $O(n + m)$ where $n = |V|$ and $m = |A|$.[1]

Balancing Procedure: An acyclic topology graph $G = (V, A, H, w)$ is balanced if and only if all paths between any given pair of nodes are of equal length and the removal of any arc in H disconnects G. A heuristic procedure, *balance*, for balancing G is presented below. It returns a set of arcs $R \subseteq A$ such that the derived topology graph $(V, A - R, H - R, w)$ is balanced.

The procedure works in a recursive manner by partitioning the topology graph into two smaller topology graphs, balancing them independently, and then merging the solutions. The partition is obtained by determining a minimum cost cutset (mincut) CS of the topology graph. The intuitive idea behind using a mincut is to minimize the sensitivity of the overall solution to the degree of optimality of the merging process (which is based on a greedy heuristic). The mincut is determined by applying the maximum flow algorithm [11] to the topology graph, with capacity of arc $a \in A$ defined by $w(a)$.

During merging, a greedy algorithm is first used to determine a maximal set of LOAD registers taken from the mincut CS that can join the two balanced substructures while keeping the merged topology graph balanced. From the definition of B-structures, it follows that if any HOLD register is used to connect the two balanced substructures, no other register should connect them. Thus, an alternative to using the set of LOAD arcs derived above is to use the maximum-weight HOLD arc in the mincut. The costs of these two solutions are compared to determine the set of arcs to be actually used for merging.

function balance ($G = (V, A, H, w)$: acyclic topology graph): Returns $R \subseteq A$ such that $(V, A - R, H - R, w)$ is balanced.

1) If (*check*(G) = SUCCESS) then return ($R \leftarrow \phi$), else proceed.

2) CS $\leftarrow$ minimal cost cutset of G; let G_s, G_d be the subgraphs of G induced by CS.

3) Balance G_s and G_d separately; $R \leftarrow$ *balance*$(G_s) \cup$ *balance*$(G_d) \cup$ CS.

4) Let $\mathrm{CS}_H \leftarrow \mathrm{CS} \cap H$, $\mathrm{CS}_L \leftarrow \mathrm{CS} \cap (A - H)$ be a partition of CS into its HOLD and LOAD registers. Sort the arcs in CS_L in order of decreasing cost.

5) $\mathrm{CS}'_L \leftarrow \phi$, the set of LOAD arcs retained in the topology graph when merging G_s and G_d.

6) For all arcs a in CS_L, in order of decreasing cost:
Check whether the inclusion of a makes the merged graph unbalanced:
If [*check*$(V, (A - R) \cup \mathrm{CS}'_L \cup \{a\}, H - \mathrm{CS}, w)$ = SUCCESS]
then $\mathrm{CS}'_L \leftarrow \mathrm{CS}'_L \cup \{a\}$.

7) Let $a_H \leftarrow$ highest-cost arc in CS_H.

8) If $\sum_{a_L \in \mathrm{CS}'_L} w(a_L) > w(a_H)$ then $R \leftarrow R - \mathrm{CS}'_L$ else $R \leftarrow R - \{a_H\}$.

9) Return R. □

The run time of the above algorithm consists of two parts: a) the computation of a minimum cutset at each recursive step, which may be repeated $O(m)$ times where $m = |A|$; and b) the invocation of the *check* procedure, which is carried out at most once for every arc in a minimum cutset, i.e., $O(m)$ times. Since step a) dominates the time required by the *check* procedure, the overall complexity is that of computing $O(m)$ minimum cutsets.

In this section, we have described a procedure for determining a near-minimal cost set of registers to be made scannable. Since a branch-and-bound algorithm is used to solve a part of the problem, the time complexity of the procedure grows exponentially in the worst case. However, the computation time is not excessive since the procedure deals with each cloud of connected combinational logic as a single node irrespective of the number of gates/transistors within the cloud.

Fig. 5(a) shows a topology graph that is not balanced. (HOLD registers are indicated by bold arcs.) Fig. 5(b) shows the result of applying *balance* to it; the arcs removed are (e, f) and (g, i), and the registers in the circuit that correspond to these arcs must be included in the scan path.

VI. Implementation of Scan Path

The BALLAST test application procedure, in its simplest form, consists of scanning a pattern into the selected registers and holding it constant for d clock cycles, where d is the depth of the B-structure comprising the kernel. If the overhead of providing all selected SPR's with HOLD modes is acceptable, no extra work needs to be done. However, not all patterns need to be held at the kernel inputs during all d clock cycles. In this section, we show how by introducing

[1] This is a tighter bound than originally published and we thank the anonymous reviewer who pointed it out.

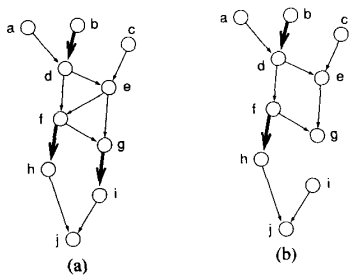

Fig. 5. Illustration of balance procedure. (a) Original topology graph. (b) Balanced topology graph.

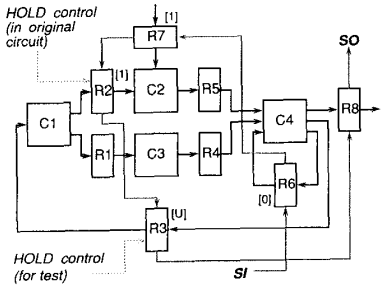

Fig. 6. Example of scan path implementation.

dummy bits the need for some SPR's to have a HOLD mode can be eliminated.

Example

Consider the BALLAST partial scan design shown in Fig. 1(b). The scan path contains the registers $R6$ and $R3$, which are ordered in the scan path as shown. Each test pattern consists of two parts, t_3 and t_6, corresponding to the SPR's $R3$ and $R6$, respectively. If $R3$ and $R6$ are both provided with HOLD modes, one way to apply the test is to scan in the combined pattern t_3t_6 and hold it in the scan path for two additional clock cycles before loading the output of $C4$ into $R6$ and $R3$ after the third clock cycle.

The test result loaded back into the scan path depends on the value in $R3$ during the first clock cycle and on the value in $R6$ during the third clock cycle. Hence, if we can arrange to have t_3 in $R3$ in the first clock cycle and t_6 in $R6$ in the third clock cycle, the HOLD modes in these registers can be eliminated without affecting the test result. This can be done by inserting two dummy (*"don't care"*) bits in the test pattern between t_3 and t_6. When t_3 becomes available in $R3$, t_6 is still displaced by 2 bits from its final destination in $R6$. After two more shifts, during the third cycle $R6$ contains the correct input pattern. Hence, the desired test result is loaded into $R3$ and $R6$ after the third clock cycle. Thus, the modified test pattern with two dummy bits eliminates the need for HOLD modes in both $R3$ and $R6$.

The example presented above is a simple illustration of the concept used to minimize the overhead due to additional HOLD modes. In general, however, it may not be possible to eliminate the HOLD modes of all the SPR's. We now describe how the number of additional HOLD modes can be minimized in an arbitrary circuit.

Construction of Scan Path

In general, a SPR can play two roles: it can serve as a *driver*, i.e., a SPR that feeds test inputs to a cloud, or as a *receiver*, i.e., a SPR that is fed test results by a cloud. A single SPR can simultaneously play both roles; it can drive test patterns into one cloud and receive test results from another. An example of a pure driver or pure receiver would be a boundary scan element in a chip employing the boundary scan methodology.

Let the *distance* between a driver and a receiver, given that at least one path exists between them, denote the number of registers in any path between the driver and the receiver (excluding themselves) through the kernel. Note that since the kernel is a B-structure, all such paths must have the same number of registers. If the distances from a given driver to all receivers to which it has a path (including itself if it is also a receiver) have the same value, define the *latency* of the driver as this value; otherwise, let the latency be undefined. A latency of x for a driver implies that the test result loaded into the scan path during clock cycle k depends only on the value present in the driver during clock cycle $k - 1 - x$.

The above concepts are illustrated in Fig. 6. The registers $R2$, $R3$, $R6$, $R7$, and $R8$ are included in the scan path and ensure that the kernel is a valid B-structure. Note that $R7$ and $R8$ are not strictly required to be scannable to ensure that the kernel is a B-structure; however, they may be included in the scan path if the circuit employs boundary scan. In this circuit, $R7$ is a pure driver; $R8$ is a pure receiver; and $R2$, $R3$, and $R6$ play both roles. In the original circuit, all registers are LOAD registers except $R2$, which has a HOLD mode. The latency of each driver in Fig. 6 is indicated alongside it as $[x]$. Undefined latencies are denoted by $[U]$.

We now need to determine the following: 1) which SPR's (if any) should be provided with a new HOLD mode; 2) an ordering of all the SPR's in the scan path; and 3) a pattern of dummy bits to be inserted into each test pattern. We consider these issues in turn in the following rules for constructing the scan path.

Given a driver with *undefined* latency, the test result due to each test pattern depends on a single pattern being present in the driver during more than one clock cycle. This follows from the definition of latency. For example, consider the driver $R3$ in Fig. 6. If the scan path is to be loaded with the test result during clock cycle k, the result to be loaded into $R2$ depends on the pattern in $R3$ during $k - 1$, and the results to be loaded into $R8$, $R6$, and $R3$ depend on the pattern in $R3$ during $k - 3$. For single-pattern testability it is essential that the same desired value be present in $R3$ during $k - 1$ as well as $k - 3$. Hence, $R3$ must be provided with a HOLD mode. The general rule is stated below.

Rule 1: All drivers whose latencies are undefined must have HOLD modes.

The scan path is constructed by forming $d + 2$ groups of SPR's, where d is the depth of the B-structure comprising the kernel. Fig. 7 shows the sequence of groups in the scan path. The group closest to the scan-in pin is Group 0 and the group closest to the scan-out pin is Group $d + 1$. Within each group the ordering of SPR's may be based on routing considerations. SPR's are assigned to groups according to the three rules presented below.

Since pure receivers never need to be supplied with test input data, all such registers can be placed at the scan-out end of the scan path so as to minimize the number of clock cycles needed to shift a new pattern into the scan path. In the example of Fig. 6 (which has $d = 2$), $R8$ is placed in Group 3 of the scan path. This principle is stated in the following rule.

Rule 2: All pure receivers are placed in group $d + 1$.

Drivers are placed in the scan path according to their latency values. As we have seen, a driver with a well-defined latency need not have a HOLD mode, provided the arrival time of the desired test data in this driver is synchronized with the appropriate arrival times of test data in the other drivers. For example, in Fig. 6, $R6$ and $R7$ are not provided with HOLD modes. For the test to be applied correctly to the kernel, $R6$ must receive its correct pattern exactly one clock cycle after $R7$, since the difference in their latency values is 1.

One way to achieve this synchronization is as follows. Drivers with high latency are placed further along the scan path. The test patterns for all drivers are concatenated into a single pattern and scanned in; however, dummy bits are placed at selected positions in the test pattern so that test data destined for drivers with lower latencies lag behind those for drivers with higher latencies. The following two rules deal with the ordering of registers so that the test patterns can be applied in this manner, and the actual formatting of pattern is described in a final rule.

Fig. 7. Organization of scan path.

Rule 3: All drivers not having a HOLD mode and having latency l must be placed in group l.

Note that all drivers not having a HOLD mode must have well-defined latency values. This rule ensures that drivers having higher latency are placed further along the scan path than those with lower latencies. Thus, drivers in group d receive the correct pattern first; at this instant, the testing of the kernel effectively begins even though the patterns destined for drivers in Group 0 still lag by d clock cycles.

So far we have not considered the placement of drivers having HOLD modes. The following rule places such drivers in the same group as drivers having latency d. This ensures that they are among the first drivers to receive their correct pattern, which they then hold for the duration of the test (i.e., for the next d clock cycles).

Rule 4: All drivers provided with HOLD modes for test and all drivers with HOLD modes already present are placed in group d.

Thus, both Rules 3 and 4 may assign drivers to Group d. According to these rules, $R2$ and $R3$ are placed in group 2, $R7$ is placed in Group 1, and $R6$ is placed in Group 0. The resulting scan path is shown in Fig. 6.

Test Application

Once the drivers have been ordered according to the grouping determined above, we simply need to ensure that the pattern for each group of drivers lags behind the pattern for its neighboring group by one clock cycle. This is done by the following rule.

Rule 5: Each single-pattern test is modified by introducing one dummy bit between each pair of consecutive groups i and $i + 1$, $0 \leq i \leq d - 1$.

Thus, a total of d dummy bits is added. Two or more adjacent dummy bits may be present in a sequence when there are empty groups in the scan path. In the example of Fig. 6, if t_i represents an input pattern for register Ri and $\times$ represents a dummy bit, the modified test pattern to be scanned in (ordered from left to right) is $t_3 t_2 \times t_7 \times t_6$. No input pattern is required for $R8$ since it is a pure receiver.

When the scan path is implemented as described in this section, the formal test procedure given in Section III needs to be modified by generalizing step 2a) as follows.

2 a) Place all scan registers having a HOLD mode in the HOLD mode, all other scan registers in the SHIFT mode, and all nonscan registers in the LOAD mode for d clock cycles. (This allows test data to propagate through the kernel.)

Circuit Modifications

In addition to constructing a scan path and providing HOLD modes to scan registers having an undefined latency, some modifications may be required to the control signals for HOLD modes of both scan and nonscan registers. Different HOLD registers may share the same control lines; however, in order to ensure that the various steps in the test plan can be executed, the following two requirements must be satisfied. First, no HOLD control signal may feed both a scan register and a nonscan register; second, all such control signals must be controllable through external I/O pins. The result is a small logic overhead, and possibly additional I/O pins, for ensuring that the HOLD signals are appropriately configured. Note that during the single-pattern testing of the kernel, the HOLD modes of only the scan paths registers get exercised. Details of the circuit modifications, as well as aspects of testing HOLD modes of nonscan registers, are provided in [8].

VII. Case Study

Both full scan and the BALLAST technique were applied to a Viterbi decoder designed by the Jet Propulsion Laboratory. The basic building block was a chip containing 16 butterfly circuits. Each chip contained 448 FF's and the total gate count was 7280. In our overhead analysis, we have ignored the area due to 16 shift registers present on the chip since they do not cause an area overhead for either full or partial scan. In both versions, a global test mode signal was used to control the operation of the scan path FF's. The various characteristics of both designs are shown in Table I. The logic overhead figures are based on the increase in the gate count.

TABLE I
Comparison of Full Scan and BALLAST Partial Scan

Type of scan	Full scan	BALLAST
No. of scan path FFs	448	256
Logic overhead	24.6%	14.1%
Pin overhead	3	4
No. of test patterns	34	41
Clock cycles to apply each pattern	449	262
Overall test time	15,714	10,998

BALLAST required only 256 of the 448 FF's to be made scannable, of which only 96 were required to hold data during test application. This was achieved in the following manner. Instead of one global clock signal controlling all FF's in the circuit, two separate clock signals CK1 and CK2 were used. CK2 was connected to the 96 scan FF's that needed to hold data during test; CK1 was connected to all other scan and nonscan FF's. Both clocks were operated simultaneously at all times *except* during the part of the test plan when the 96 scan FF's connected to CK2 were required to hold their data. During these clock cycles, only CK1 was operated while CK2 remained inactive. This implementation required one additional I/O pin and there would be an additional routing overhead due to the separate clock signals.

The resulting kernel had depth 5. The number of single-pattern tests generated by combinational ATPG on the combinational equivalent circuit was higher than in the full scan case. However, there was a 30% reduction in the overall test time for the BALLAST circuit due to the shorter scan path, even though dummy bits were added to each test pattern.

VIII. Conclusion

In this paper, we have described a methodology for partial scan design. The foundation of this approach is based on defining a new class of synchronous sequential circuits called *balanced structures*. They have the following properties which make them useful as kernels in a partial scan circuit: 1) they are single-pattern testable for all detectable faults in the combinational logic and nearly all faults in the storage elements, 2) the FF's internal to these networks need not be made scannable, and 3) they can be treated as combinational circuits for the purpose of test generation.

The concept of balanced structures can be used to reduce various overheads in partial scan design for arbitrary sequential circuits. By identifying the balanced substructure of the circuit that has the largest number of FF's, it is possible to minimize the test overhead by configuring the balanced substructure as a kernel to be tested using a partial scan path without any loss in fault coverage. The computation time for this analysis is in general orders of magnitude lower than the time required for sequential ATPG on the original circuit. Some FF's in the scan path may need to have a HOLD mode so that single-pattern tests can be applied to the kernel. By ordering the scan path registers appropriately, the number of such HOLD modes required can be reduced to a minimum, provided the test patterns are modi-

fied by inserting dummy bits where necessary. Case studies indicate that the logic overhead for the scan path can be reduced significantly using this partial scan methodology, particularly in pipelined circuits such as those which often occur in digital signal processing chips. By eliminating scan registers in the critical path, the performance of the circuit can also be enhanced.

References

[1] E. Trischler, "Design for testability using incomplete scan path and testability analysis," *Siemens Forsch.- u. Entwickl.-Ber.*, vol. 13, no. 2, pp. 56–61, 1984.

[2] H.-K. T. Ma, S. Devadas, A. R. Newton, and A. Sangiovanni-Vincentelli, "An incomplete scan design approach to test generation for sequential machines," in *Proc. IEEE Int. Test Conf.*, Sept. 1988, pp. 730–734.

[3] K.-T. Cheng and V. D. Agrawal, "An economical scan design for sequential logic test generation," in *Proc. Fault-Tolerant Comput. Symp. (FTCS-19)*, June 1989, pp. 28–35.

[4] V. D. Agrawal, K.-T. Cheng, D. D. Johnson, and T. Lin, "A complete solution to the partial scan problem," in *Proc. IEEE Int. Test Conf.*, 1987, pp. 44–51.

[5] A. Kunzmann, "Produktionstest synchroner Schaltwerke auf der Basis von Pipeline-strukturen," in *Proc. 18. Jahrestagung der Gesellschaft für Informatik*, Hamburg, 1988, pp. 92–105, Informatik-Fachberichte 188, Springer-Verlag.

[6] A. Miczo, *Digital Logic Testing and Simulation*. New York: Harper and Row, 1986.

[7] R. Gupta, R.Gupta, and M. A. Breuer, "BALLAST: A methodology for partial scan design," in *Proc. Fault-Tolerant Comput. Symp. (FTCS-19)*, June 1989, pp. 118–125.

[8] —, "A methodology for partial scan design using balanced sequential structures," Tech. Rep. CRI-88-59, Univ. of Southern California, Dep. Elec. Eng.-Syst., Dec. 1988.

[9] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. San Francisco, CA: Freeman, 1979.

[10] H. W. Lenstra, "The acyclic subgraph problem," Tech. Rep. BW 26/73, Mathematical Centre, 49, 2e Boerhaavestraat, Amsterdam, July 1973.

[11] R. E. Tarjan, *Data Structures and Network Algorithms*. Philadelphia, PA: SIAM, 1983.

A Partial Scan Method for Sequential Circuits with Feedback

KWANG-TING CHENG AND VISHWANI D. AGRAWAL

Abstract—**This paper presents a method of partial scan design in which the selection of scan flip-flops is aimed at breaking up the cyclic structure of the circuit. Experimental data are given to show that the test generation complexity may grow exponentially with the length of cycles in the circuit. This complexity grows only linearly with the sequential depth. Graph-theoretic algorithms are presented to select a minimal set of flip-flops for eliminating cycles and for reducing the sequential depth. Tests for the resulting circuit are efficiently generated by a sequential logic test generator. An independent control of the scan clock allows insertion of scan sequences within the vector sequence produced by the test generator. We obtained a 98% fault coverage for a 5000 gate circuit by scanning just 5% of the flip-flops. A novel design of scan flip-flop reduces the input pin and signal routing overheads of the added scan clock.**

Index Terms—**Design for testability, scan design, sequential circuit testing, test generation.**

Manuscript received July 6, 1989; revised November 18, 1989. An earlier version of this paper appeared in the Proceedings of the 19th International Symposium on Fault-Tolerant Computing under the title, "An Economical Scan Design for Sequential Logic Test Generation."

The authors are with AT&T Bell Laboratories, Murray Hill, NJ 07974.

IEEE Log Number 8933873.

Introduction

Partial scan is an attractive method of designing VLSI circuits with guaranteed testability. Two approaches have been used. In the first approach [1], scan flip-flops are selected by testability analysis. A sequential logic test generator is then used to obtain test vectors. The scan overhead and fault coverage depend upon how well the testability analysis represents the problems of the test generator. The second approach [2] uses functional vectors and a combinational test generator to select scan flip-flops and to generate test vectors only for the faults not detected by the functional vectors. The scan overhead, in this case, may depend on the quality of functional vectors.

The motivation in the present work is to eliminate the reliance on testability analysis or functional vectors. We observed the performance of a sequential logic test generator that uses time-frame expansion of the circuit to generate tests by the *D*-algorithm. We found that the cycles in the circuit graph, where nodes are the flip-flops and the edges are the combinational signal dependencies, are mainly responsible for the test generation complexity. In the absence of such cycles, even large sequential depth causes no serious problem to the test generator.

Finite state machines that can be realized without a cyclic structure are known as *definite machines* [3]. These circuits behave like a pipeline and their test generation complexity is similar to that of a combinational circuit [4]. In general, however, a sequential circuit may have cycles for several reasons; its function may be nondefinite or cycle-free implementation may require too many extra gates and flip-flops. The main idea we present in this paper is that of breaking some or all cycles by scanning a minimal subset of flip-flops.

We use a graph-theoretic method to select a minimal set of flip-flops for scan. The clock of scan flip-flops is given a separate control to allow insertion of scan sequences between the vector sequences produced by the test generator. We also give an efficient flip-flop design for controlling the scan clock without any extra pin over what is needed for the normal scan design. The results show that for a large circuit, by scanning only 5% of the flip-flops, tests could be effectively generated automatically and without any interaction from the designer.

What Makes Test Generation Difficult?

The difficulty in generating tests for general sequential circuits arises due to the poor controllability and observability of memory elements. In practice, the level of this difficulty is circuit-dependent. Some circuits can be easily handled by a test generator while others take enormous computing effort. Table I shows the test generation results obtained from a sequential logic test generation program [5]. One circuit is a traffic light controller (TLC) and the other (CHIP-A) is a CMOS chip designed for a graphics terminal. Even though CHIP-A is three times larger than TLC, our test generator processed it in one-fifth the time and produced a significantly higher fault coverage. The CPU time in this table and for all other results given in this paper is for a VAX8650 computer.

The results in Table I show that the gate count is definitely not a key parameter in determining the complexity of test generation. Sequential depth which is often cited as a *measure of sequentiality* of a circuit does not explain the result either. If we define *sequential depth* as the largest number of flip-flops on a path between inputs and outputs, then it is 14 for both circuits. However, if we consider the *cycles* formed by flip-flops, CHIP-A contains no cycles greater than length one. A length one cycle (or loop) is formed when a flip-flop's output feeds back into its own input after passing only through combinational logic. TLC, on the other hand, has many longer cycles that may be responsible for the higher complexity.

Test Length and Cycles in Sequential Logic

Consider a directed graph $G = \{V, E\}$ of a synchronous sequential circuit. A vertex v_i in this graph represents a flip-flop (or state