Short Papers

An Efficient Scan Tree Design for Compact Test Pattern Set

Shibaji Banerjee, Dipanwita Roy Chowdhury, and Bhargab B. Bhattacharya

***Abstract*—Tree-based scan path architectures have recently been suggested for reducing test application time or test data volume in today's high-density very large scale integrated circuits. However, these techniques strongly rely on the existence of a large number of compatible sets of flip-flops under the given test set and therefore may not be suitable for a highly compact test set generated by an efficient automatic test pattern generator tool. Tree-based architectures also suffer from loss of fault coverage while achieving a significant reduction ratio for test time or data. In this paper, to circumvent this problem, a new two-pass hybrid method is proposed to design an efficient scan tree architecture based on approximate compatibility. The method is particularly suitable for a highly compact test set having fewer don't cares and low compatibility. Finally, to reduce the volume of scan-out data, test responses shifted out from the leaf nodes of the scan tree are compacted by a space compactor, which is designed specially for the proposed scan tree architecture. The compactor uses an XOR tree, and its overhead is low. The design thus offers a solution to both test data and response compaction. Experimental results on various benchmark circuits demonstrate that the proposed algorithm outperforms the earlier methods in reducing test application time significantly without degrading fault coverage.**

***Index Terms*—Design-for-testability, scan path, space compaction, stuck-at faults, testing, very large scale integration.**

I. Introduction

Controllability and observability of a digital circuit can be increased by various well-known design-for-testability techniques. Among them, the most popular one is the widely used serial full-scan methodology, which transforms a sequential circuit into its combinational parts in the test mode. Although the full-scan method reduces the cost of test generation and provides high fault coverage, the inherent serial nature of the scan path increases the test application time and energy consumption in test mode significantly. The number of clock cycles needed to scan in/out the test data is equal to the product of the number of test patterns and the length of the scan chain. Hence, the test application time (also test data volume), and consequently the usage cost of the automatic test equipment, can be lessened either by reducing the number of test patterns or by shortening the scan chain length, as used in tree-based scan path architectures. A fewer number of test patterns may however reduce the fault coverage. On the other hand, a compact test set with high fault coverage has relatively smaller number of don't cares in the test patterns, which causes more incompatibility relationships among the flip-flops. This in turn, makes the scan tree design inefficient, i.e., it may be hard to find a tree of reasonably low depth. As a result of these two key factors being conflicting in nature, the design of a scan tree that targets to achieve a good reduction ratio of test time/data and admits high fault coverage, becomes a hard problem to solve. Moreover, a low-depth scan tree, which is desirable from the viewpoint of time/data reduction, is likely to have a larger number of scan outputs, which results an increase in cost of collecting response data. Thus, a space compactor may be needed to reduce the width of scan-out data either for direct observation or before feeding them to a multiple-input shift register (MISR). Hence, in a scan tree architecture, the problems of test data/time reduction and response compaction assume greater significance than those in a conventional serial chain.

There are several existing methods that can be employed to tackle the problem of reducing test application time. One approach is to configure the scan elements into multiple scan chains [1], [2] at the cost of having an increased number of scan-in and scan-out pins. Hybrid test generation techniques [3], [4] are computationally expensive and are not suitable for large sequential circuits. A scan-based built-in self-test is introduced in [5], where a scan chain is partitioned into multiple segments by inserting XOR gates between two adjacent scan segments. The test responses of each scan segment can be observed through an XOR tree. However, this technique does not address the problem of testing embedded cores used in system-on-a-chip design [6].

The Illinois Scan Architecture (ILS) has been recently proposed to circumvent this problem [7]–[9]. The ILS is shown to be useful for both the standalone chip and embedded cores. It does not require any additional test pin other than the ones used in full scan. In the ILS scan, several branches of the scan paths emanate from a single scan-in pin of the circuit, which may however cause overloading. An alternative approach called *scan tree* has been proposed [10]–[14] to reduce the test application time or the test data volume. In such a scan architecture, the structure resembles a tree, where one cell drives the successor scan cells as in a tree. The flip-flops corresponding to the leaf nodes of the scan tree are fed to an MISR for response collection and analysis. A scan tree also requires a single scan-in pin as in full scan or ILS. The same test data bit, however, arrives in the scan cells, which have an equidistant common ancestor cell. Thus, in a scan tree architecture, all the bits in each of the test vectors that are to be loaded in the scan cells (flip-flops) lying at the same depth of the tree must be compatible (i.e., nonconflicting) among themselves. If the above condition is satisfied, the corresponding scan cells are also called compatible under the given test set. The effectiveness of the scan tree depends on the correlation among the test data of different scan cells. The test application time in a scan tree depends on the number of test patterns and the depth or height (i.e., length of the longest path in the tree) of the tree. The latter strongly depends on the compatibility of the flip-flops under the given test set. Greater compatibility among the flip-flops is likely to reduce the depth of the scan tree. Most of the previous works on generating a scan tree used noncompact test patterns (containing many don't cares) in order to obtain a large number of compatible relationships. In these cases, the number of test patterns considered is usually large (compared to those generated by an optimized automatic test pattern generator (ATPG) tool achieving the same fault coverage). The method used by Miyase and Kajihara [10] may also handle a compact test set but requires iterative modification of the test vectors to enhance compatibility among the flip-flops while generating the scan tree. However, this method may also achieve decent reduction rate without any iteration. But all these methods may not yield a scan tree of low depth for a highly optimal compact test set, where the number of don't cares is few, and/or the compatibility relationships among them are infrequent.

Manuscript received November 23, 2005; revised April 21, 2006 and June 28, 2006. This paper was recommended by Associate Editor S. Hellebrand. This paper was presented in part at the International Conference on VLSI Design, Hyderabad, India, January 3–7, 2006.

The authors are with the Department of Computer Science and Engineering, Indian Institute of Technology, Kharagpur 721 302, India (e-mail: shibaji@vlsi.iitkgp.ernet.in; drc@cse.iitkgp.ernet.in; bhargab@isical.ac.in).

Digital Object Identifier 10.1109/TCAD.2007.895840

0278-0070/$25.00 © 2007 IEEE

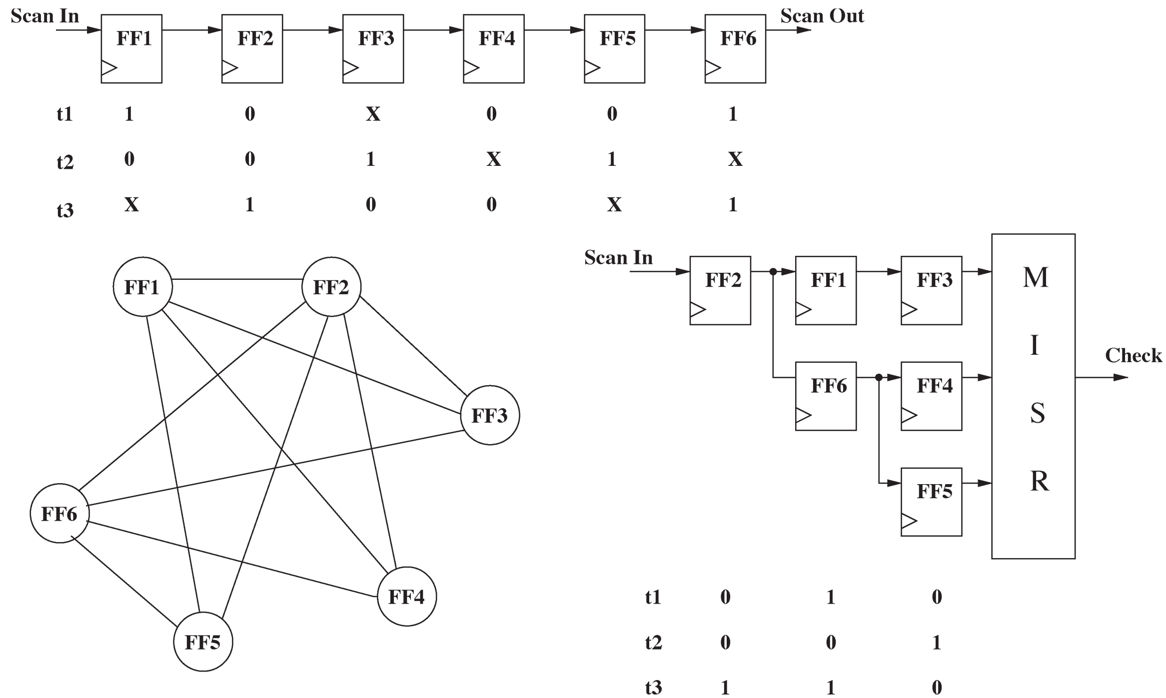

Fig. 1. Designing the scan tree architecture.

Thus, they cannot exploit, while designing the scan tree, the advantage of using such a test set of small volume without causing significant degradation of fault coverage.

In this paper, a new graph-based two-pass algorithm is proposed to design the scan tree architecture. The present algorithm is particularly applicable for a compact test set, which is typically generated by commercial ATPG tools, where the compatibility among the flip-flops is usually low. First, the compatibility relationships among the flip-flops are explored for a given test set, and a scan tree architecture is designed with minimal incompatibility. Next, the same ATPG tool is run again to generate a new test set satisfying the logical constraints on the secondary inputs imposed by the scan tree designed above. These vectors will be applied in the tree (i.e., broadcast) mode. To cover the remaining hard-to-detect (HTD) faults, if any, a few test vectors are chosen from the original test set for application in the serial mode. Finally, to reduce the cost of response analysis, a low-overhead space compactor is specially designed using an XOR tree to compact the set of leaf nodes of the scan tree. Experiments on various benchmarks reveal very encouraging results in terms of tree depth, test time reduction, fault coverage, and compactor overhead, which indicate that the proposed hybrid method based on approximate compatibility works very well for scan tree design.

II. Scan Tree Architecture

Several tree-based scan architectures have been proposed [11], [12] earlier to reduce the test application time. Fig. 1 presents an example of a single scan chain consisting of six scan cells. The test vectors are shown in Fig. 1. It is evident that for these test vectors, the scan cells $FF1$ and $FF6$ are *compatible* as the column vectors $[10X]$ and $[1X1]$ corresponding to them (shown in Fig. 1), where X denotes a don't care, are nonconflicting. A set of scan cells is said to be compatible if every pair of scan cells in the set is compatible under the given test set. Thus, the set $\{FF3, FF4, FF5\}$ is compatible. The design of a scan tree is based on the compatibility of scan cells. In the above example, the groups of compatible scan cells are $\{FF2\}$, $\{FF1, FF6\}$, and $\{FF3, FF4, FF5\}$. As $FF2$ is incompatible with all other scan cells, it cannot be grouped with other scan cells. The corresponding tree design is shown in Fig. 1, where the group of scan cells belonging to the same level of the tree must be compatible so that they receive the same test data bit during shift-in.

To determine the compatible scan cells from the test set, an incompatibility graph [11] is constructed from the test set. In the incompatibility graph, a vertex corresponds to a scan cell, and an undirected edge between two vertices exists if and only if the two scan cells are incompatible, i.e., if the corresponding column vectors (see Fig. 1) have conflicting bits (0 and 1) for at least one test pattern.

In Fig. 1, there are six vertices in the incompatibility graph. The grouping of scan cells to construct the tree architecture is determined following the chromatic partitioning of the incompatibility graph.

As mentioned earlier, a scan tree architecture will be meaningful only when a large number of flip-flops are compatible. However, for a compact test set with a fewer number of don't cares, the probability of having compatible flip-flops reduces and the situation becomes worse when the number of flip-flops is large. One solution to the problem is to choose an arbitrary scan tree structure *a priori* and then use a constrained ATPG to generate the test patterns for the circuit. The disadvantage of the method is that the number of test patterns will be large and the fault coverage will be low. Another solution (known as hybrid method) is to select a subset of the test set so that a scan tree of reasonable depth is constructed and to apply the remaining vectors in the serial mode [11]. However, this does not fare well for a highly compact test set. The present paper solves this problem by adopting a two-pass hybrid method.

III. Proposed Algorithm for Scan Tree Design

In this section, the theoretical formulation of the method is first presented followed by a description of the proposed algorithm.

A. Scan Tree Organization

We start with a given compact test set (T_n) of the circuit under the single stuck-at fault model and analyze the compatibility relationships

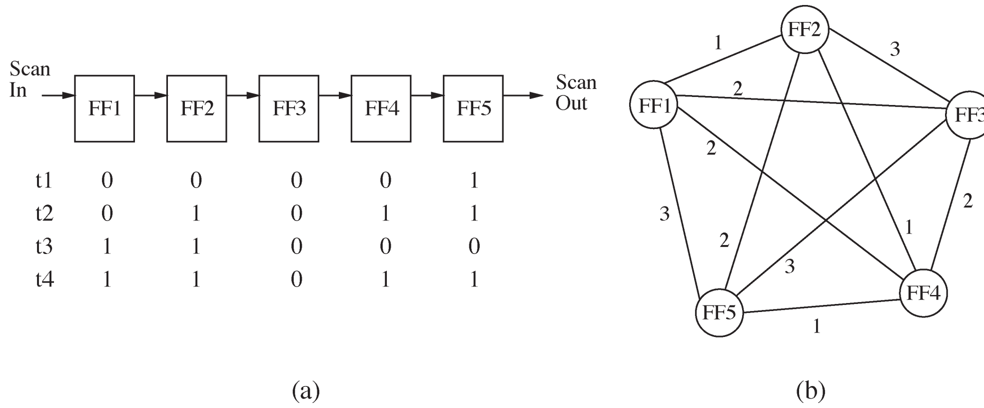

Fig. 2. Test vectors (T_n) and the weighted graph.

among the flip-flops. Typically, the test set generated by efficient commercial ATPG tools consists of very few don't cares, which results in more incompatibility among the flip-flops.

For example, consider the test set shown in Fig. 2, which consists of four five-bit test vectors. The linear scan chain has five flip-flops. According to the above description, flip-flop 1, which is denoted as $FF1$, corresponds to bit 1 in each test vector, flip-flop 2, which is denoted as $FF2$, corresponds to bit 2, and so on [Fig. 2(a)]. In this example, no pair of flip-flop is compatible, and hence, the scan tree will reduce to a single linear chain.

We define the *incompatibility distance* (which is denoted as d) between two flip-flops as the number of conflicting bits in the two corresponding column vectors when the test vectors are arranged in rows. For the example in Fig. 2, we have the following distance values:

$$\begin{aligned} d(FF1, FF2) &= 1, \quad d(FF1, FF3) = 2 \\ d(FF1, FF4) &= 2, \quad d(FF1, FF5) = 3 \\ d(FF2, FF3) &= 3, \quad d(FF2, FF4) = 1 \\ d(FF2, FF5) &= 2, \quad d(FF3, FF4) = 2 \\ d(FF3, FF5) &= 3, \quad d(FF4, FF5) = 1. \end{aligned}$$

A complete undirected graph $G(V, E)$, which is called *weighted incompatibility graph* (WIG), is now constructed with the flip-flops as vertices and these distance values as weights on the corresponding edges. A zero-weight edge between two vertices denotes a compatible pair of flip-flops. As mentioned before, in Fig. 2, there is no pair of compatible flip-flops.

To construct the scan tree based on minimum incompatibility, we proceed as follows. We define a w-*distance* WIG as the largest subgraph of WIG $G(V, E)$ such that all the edges present in the subgraph have weight w. Clearly, given w, the subgraph is unique. We extract the subgraph with the smallest w value and continue to process subgraphs with higher w values in steps during the construction of the scan tree. The following steps illustrate the process.

1) Extract the largest subgraph $G'(V', E')$ of $G(V, E)$ by extracting all the edges E' of weight w from G, and let V' represent the set of corresponding vertices connected by the edges in E'.
2) Obtain the complement of the graph G'. Given a undirected graph $G' = (V', E')$, the complement of G' is defined as $\overline{G'} = (V', \overline{E'})$, where $\overline{E'} = \{(u', v') : u', v' \in V', u' \neq v'$, and $(u', v') \notin E'\}$.
3) In $\overline{G'}$, determine the minimum number of colors needed to color the graph. Based on this chromatic partition, group the vertices of $\overline{G'}$, i.e., the corresponding flip-flops into nearly compatible groups. Thus, the flip-flops grouped in this fashion will have incompatibility distance w among themselves.

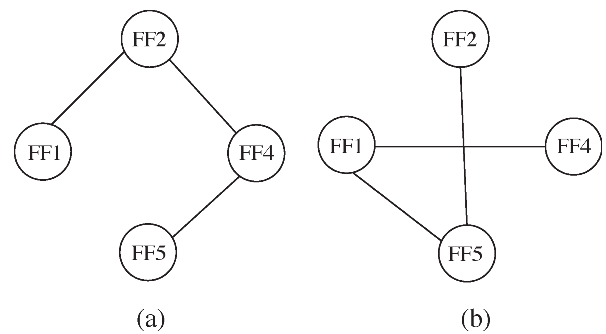

Fig. 3. (a) Subgraph $G'(V', E')$ and (b) its complement $\overline{G'} = (V', \overline{E'})$.

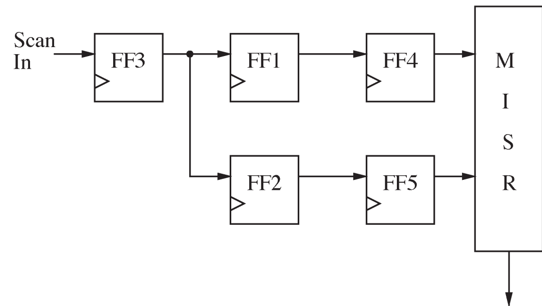

Fig. 4. Scan tree structure of Fig. 2(a).

The above procedure based on chromatic partitioning of the complementary graph $\overline{G'}$ is essentially equivalent to the *minimum clique partitioning* of the graph G. Since the decision version of this problem is known to be *NP*-complete [15], we have used a fast greedy heuristic for chromatic partitioning [16]. Other improved heuristic techniques for coloring may also be used for more accurate results [17]. For example, the graph $G(V, E)$ shown in Fig. 2 can be used to find the groups of flip-flops when the incompatibility distance is 1. For this purpose, a subgraph $G'(V', E')$ is constructed from G, where the weight of each edge in G' is 1, and V' are those vertices that are connected by E' [as shown in Fig. 3(a)]. The complement of the graph G' is constructed as shown in Fig. 3(b). In $\overline{G'}(V', \overline{E'})$ (complement of G'), $|V'| = 4$, and $|\overline{E'}| = 3$. The vertices (flip-flops) are grouped via coloring process of the graph. The graph shown in Fig. 3(b) has no edge between $FF1$ and $FF2$. So, they are grouped. Similarly, $\{FF4, FF5\}$ are grouped. The scan tree thus obtained is shown in Fig. 4. The scan tree in Fig. 4 is formed where the bit difference between the flip-flops in each group is allowed to be 1. The process is repeated for the other higher values of w until the WIG is exhausted.

Procedure_1: Scan tree generation

Input to the Procedure_1:

(a). Total number of flip-flops (n_f)
(b). Complete set of test patterns (T_n)

1. Generate the WIG G based on the information given in a and b;
2. Set integer i to 0;
3. Find all the edges (E_i) in G having weight i;
4. Extract the largest subgraph ($G_i(V_i, E_i)$) of G, which contains the edges E_i and all the vertices V_i connected by the edges in E_i;
5. Draw the Complement $\overline{G_i}$ of the subgraph G_i;
6. Color $\overline{G_i}$ using a greedy heuristic [16] to find the groups of flip-flops; consider only those groups that have more than one flip-flop;
7. From G, delete all the vertices already grouped, and the edges incident on them;
8. Set $i = i + 1$;
9. If some edges are still present in G, go to step 3;
10. For each of the isolated vertices, if any, construct a one-member group comprising the corresponding flip-flop;
11. Arrange the groups of flip-flops in non-decreasing order of their sizes; construct the levels (one level for each group) of the scan tree following the order, starting with the scan-in pin connected to each of the flip-flops belonging to the smallest group.

Fig. 5. Procedure_1 to generate the scan tree.

Procedure_2: Generate the new test set based on ST-mode, and identify the undetectable or HTD faults

Input to the Procedure_2: The scan tree generated by Procedure_1

1. Find the complete set of collapsed faults f_c in the CUT that were detectable by the original test set (T_n);
2. Insert the constraints on the secondary inputs of the CUT imposed by the scan tree;
3. Run the same $ATPG$ tool to generate a test set T_{st} for the target faults in f_c, performing incremental fault simulation for each new vector; stop when fault coverage saturates in ST-mode; let f_d be the set of faults detected by T_{st};
4. Determine the set (f_u) of undetectable or HTD faults: $f_u = f_c - f_d$.

Fig. 6. Procedure_2 to determine the new test set and the undetectable or HTD faults in ST mode.

In this way, a scan tree is built where the flip-flops lying at the same depth of the tree will have the same incompatibility distance value among them.

B. Algorithm to Generate the Scan Tree

In this section, an algorithm is developed to design the scan tree architecture when the compatibility relations among the flip-flops are low. The algorithm consists of two procedures: Procedure_1 is used to generate the scan tree structure, based on a heuristic of graph coloring described above; and Procedure_2 is used to determine the fault coverage. A formal outline of Procedure_1 is shown in Fig. 5.

C. Test Generation Based on Scan Tree and Hybridization

Scan Tree (ST) Mode: Once we obtain a scan tree structure, we rerun the same ATPG tool to generate a new set of test vectors T_{st} with the logical constraints imposed on the secondary inputs of the circuit by the grouping of scan cells. These vectors will be applied in scan tree (broadcast) mode. The rationale behind this is as follows. Since the same ATPG tool is being run on the same circuit-under-test (CUT) with some input constraints determined by minimal incompatibility, fault coverage close to the earlier one is likely to be achieved also in the second run. We assumed single stuck-at fault model for our analysis. Procedure_2 is used for this purpose. However, in the presence of the constraints, some faults that were testable by the original test set may become untestable or HTD in the scan tree mode. Procedure_2, on termination, also identifies this hard set of faults. To achieve the same fault coverage obtained by the original test vectors in (T_n), some of them, which are appropriately chosen from the original set (T_n), can be applied to the CUT in serial scan mode (described below) as in [11] by using a simple scheme that allows to reconfigure the tree architecture into the serial mode if needed. However, in [11], given a test set, the scan tree is designed based on expanding a subset of the test set and on the resulting compatibility relations among the flip-flops. The proposed approach also differs from that described in [10], where the tree structure is iteratively modified by changing the don't care/specified bits of test vectors. For a highly compact test set, these methods may not lead to an efficient tree structure, and the actual total test application time may not reduce significantly while preserving high fault coverage. Procedure_2 is described in Fig. 6.

Serial Scan (SS) Mode: To generate the test vectors to be applied in SS mode, we consider the set of faults that were detectable by the original test set (T_n) but which have become untestable or HTD in ST mode (obtained by Procedure_2). Next, following a simple heuristic procedure, we choose a few additional vectors from the original test set (T_n) to cover the above set of hard faults for achieving the desired fault coverage in SS mode. The scan tree can be dynamically reconfigured into serial mode by employing a simple hardware. The idea is to apply the majority of test vectors in ST mode and a few in SS mode. Fig. 7 illustrates the technique. Switching of modes can be implemented by a controller, which consists of some logic and a counter that counts the number of test patterns to be applied in ST mode.

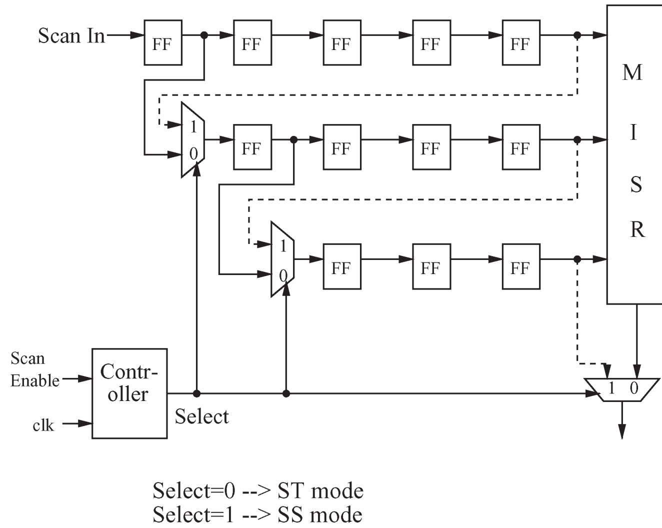

Fig. 7. Scan tree architecture with dynamic reconfiguration.

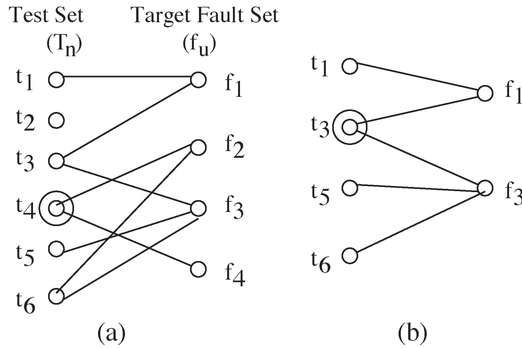

Fig. 8. Test vectors and hard faults as a bipartite graph $G(T_n, f_u)$ and its subgraph.

We start with the set of undetectable/HTD faults (f_u), which are called hard faults, and the original test patterns (T_n) from which the scan tree structure was built. The minimal number of test patterns for *SS* mode can be obtained by constructing a bipartite graph whose left set of vertices represents the vectors in the test set T_n and whose right set of vertices represents the faults in the list f_u. As shown in Fig. 8(a), $T_n = \{t_1, t_2, t_3, t_4, t_5, t_6\}$ denote the set of test patterns, and let $f_u = \{f_1, f_2, f_3, f_4\}$ denote the set of hard faults. The testability relation among the tests and detected faults can be described by a bipartite graph $G(T_n, f_u)$ [18], [19], where an edge (t, f) is given if and only if a test t detects a fault f. A fault f is said to be detected by a test set T_n if G contains at least one edge between the members of the set T_n and f.

In order to construct the bipartite graph, a complete fault simulation may be done for each pattern of T_n on the hard fault set f_u. Although the set f_u is usually small in size, to save simulation time, incremental fault simulation in forward and reverse passes may be employed to approximate the graph. An illustrative example is shown in Fig. 8(a).

Let T_s denote the set of test vectors to be applied in SS mode. Then, the problem is to find a subset T_s of T_n such that:

1) T_s covers all nodes in f_u;
2) the number of nodes in T_s is minimum.

The above problem is equivalent to the classical *minimum set cover* problem, the decision version of which is known to be *NP*-complete [15]. Thus, in this paper, a greedy approach, which terminates very fast, is used to solve the above optimization problem. The proposed heuristic is as follows. First, initialize T_s to null. We then choose the hardest fault, for example, f_h from f_u, i.e., the one having the smallest degree on the fault side of the bipartite graph; ties, if any, are resolved arbitrarily. Now, among the test vectors of T_n that have an edge to f_h, select the one with the largest degree, again resolving ties arbitrarily. The node corresponding to the vector, all the faults in f_u detected by it, and the relevant edges incident on the detected fault nodes are then deleted from the bipartite graph. In the process, the degree of some test node may also become zero. They are also deleted. The test vector selected is added to T_s, and the process is iterated until all the hard faults in f_u are covered. It is easy to show that this heuristic procedure to generate T_s will take $O(e \log e)$ time, where e denotes the number of edges in the bipartite graph.

The above algorithm can be demonstrated with the help of an example shown in Fig. 8(a). The fault f_4 has the smallest degree, i.e., hardest to detect. The (only) test pattern t_4 that detects f_4 is thus included in the set T_s. Since this test also detects f_2, the graph after node and edge deletion would look like the one shown in Fig. 8(b). Next, node f_1 (of smaller degree 2) is chosen, and between t_1 (of degree 1) and t_3 (of degree 2) that detect f_1, node t_3 (larger degree) is chosen. After this iteration, the reduced graph becomes empty, and therefore, $T_s = \{t_3, t_4\}$ will be the minimal set of test patterns that detect all the hard faults f_u.

The serial patterns thus obtained (T_s) cover all the faults that are untestable by T_{st} but testable by the original test set (T_n). However, each serial pattern requires full shifting of scan bits, and therefore, one may not include all such required patterns, thereby incurring slight loss of fault coverage. Thus, a user has the flexibility of trading off test application time/data against certain loss of fault coverage within acceptable limits.

The serial patterns may also detect some faults already detected by T_{st}. Thus, some of the vectors in T_{st} may be removed to further reduce the test application time in broadcast mode. Such vectors can be identified by fault simulation. First, fault simulation is done for the test patterns T_s on the complete fault list f_c. This is done in order to drop additional faults that are covered by these serial patterns. Let f_{ds} denote the set of all faults detected by T_s in serial mode. The fault set f_{ds} is then deleted from f_c. The remaining set denoted by f_{dst} is actually the reduced target fault list, and these faults are to be detected in ST mode. Incremental fault simulation of the vectors in (T_{st}) on f_{dst} in forward and reverse orders may eliminate some test vectors from the set (T_{st}), thereby reducing its size. The combined set of test patterns (T) is finally obtained by the union of T_s and the reduced T_{st}. Procedure_3, which is shown in Fig. 9, describes the method.

IV. Response Compaction Technique for the Scan Tree

In the scan tree architecture, scan-out is usually implemented by feeding the outputs of the flip-flops corresponding to the leaf nodes of the tree to an MISR for response collection and analysis. A scan tree of smaller depth provides better test time/data reduction, although the number of leaf nodes of the tree will tend to increase. Thus, an MISR will need to capture more scan-out bits as inputs per shift cycle, which results in larger area overhead. This problem can be solved by employing a space compactor, which will reduce the number of output bits to be fed to the MISR. For a high-ratio compactor, an MISR may be eliminated, and the compacted bits can be directly used for signature analysis. An XOR tree compactor has been recently proposed for response compaction in a scan tree architecture [20], [21]. In [20], the authors used an XOR tree connected randomly to the leaves of scan tree(s) for space compaction. Since this may cause high overhead and aliasing, a technique called *reconstructed scan forest* has been reported later to reduce aliasing and cost [21]. An example demonstrating the latter approach is shown in Fig. 10, where the six leaves of the scan tree

Procedure_3: To generate the test patterns for the circuit and to compute test application time

Input to the Procedure_3: (a). Complete set of test patterns T_n (used to generate the scan tree)
(b). Structure of the scan tree
(b). The set of test patterns T_{st} in ST-mode
(c). Set of faults f_c
(d). Set of hard faults f_u

1. Set the circuit in serial scan chain mode;
2. Generate the set of serial patterns T_s for the target faults f_u (using the heuristic covering method on bipartite graph);
3. Fault simulate incrementally for the serial patterns T_s (circuit set in SS-mode) on f_c. Let the detectable faults be f_{ds};
4. Compute $f_{dst} = f_c - f_{ds}$;
5. Fault simulate incrementally T_{st} (circuit test set in ST-mode) on the reduced target faults f_{dst}, and update T_{st} by eliminating some vectors from T_{st}, if any;
6. Compute $T = T_s \bigcup T_{st}$;
7. Compute the test application time for the complete test set T, considering T_{st} being applied in ST-mode, and T_s applied in SS-mode.

Fig. 9. Procedure_3 to generate the complete set of test patterns.

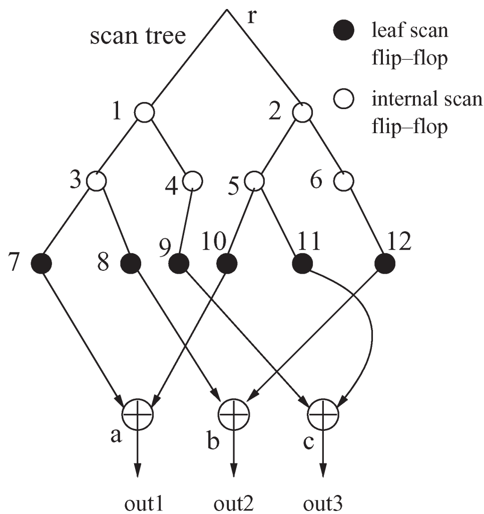

Fig. 10. Compactor proposed in [21].

are compacted to three outputs. The idea is to select the appropriate nodes for XOR-ing such that they do not have any common ancestor in the tree and thus aliasing of errors could be minimized. Further, it is desirable that the flip-flops, from which shift-out bits reach an XOR gate simultaneously, do not have a common combinational predecessor in the CUT. This strategy is adopted to prevent aliasing that may occur if the fault effect is captured by an even number of flip-flops feeding the same XOR gate of the compactor and is not propagated out along other routes, or if it is not observable at some other cycle. Other important results on compactor design and techniques for minimizing aliasing can be found in [22]–[24].

The compactor in [21] may cause aliasing when there is a flip-flop at the root node, because an error latched therein may be cancelled out at the compactor output. Further, any two flip-flops feeding the same XOR gate should not have a common ancestral internal node unless an error captured there is not propagated to other outputs. Thus, careful selection of compactor inputs is necessary for low-overhead and low-aliasing design. The scan tree produced by Procedure_1 has a special property that all the paths in the tree have equal depth. Described below is an example that illustrates how the compactor design can be greatly simplified for the type of scan trees generated by our algorithm.

Consider the scan tree shown in Fig. 11. It has 30 nodes and six leaf scan flip-flops. It consists of six branches labeled as a to f. Each node is marked by an integer, which is written within that node. Let $\{a, f\}$ be the output pair to be compacted to a single bit, where an error from the root node is to be observed. The leaf scan flip-flops of a and f are XOR-ed, and the corresponding XOR gate is denoted as XOR1 in Fig. 11. By observing output of XOR1 *(out1),* errors at the other nodes lying on the two branches a and f can be observed, except the one at the root node (single error), and those double errors affecting the pair of nodes lying at the same level on the two branches. However, instead of connecting the output of the leaf scan flip-flop of f directly to an input of XOR1, if it is connected through a two-input AND gate as shown in Fig. 11, then the above problem of aliasing can be avoided. The other output of the AND gate *(A1)* is connected to a control line "enable." By setting $enable = 0$, we can disable the branch f at XOR1, and responses reaching a can be directly observed at *out1*. During the scan-out period, the response of the root node will reach the leaf nodes of branches a and f at the end of the seventh shift cycle. At that time, we need to set $enable = 0$ to disable the branch f, so that the error latched at the root is not masked at *out1*. During other scan-out cycles, "enable" will be set to 1. To implement this feature, we need an up-counter that counts the number of levels in the scan tree. Initially, it will be reset to 0, and after each shift clock cycle, its value will be incremented by 1. After counting the highest level of the tree, it will again be reset to 0. For example, in Fig. 11, we need a mod-7 up-counter that counts from 0 to 6. When the value of counter becomes 6, the enable signal is set to 0.

After that, the counter will be initialized to 0, and the enable signal will be at logic 1. Thus, the single error at the root node will not be aliased at the compactor output.

To complete the design of the compactor, one may now choose two other leaves (for example, $\{b, e\}$) and connect the outputs of the two corresponding flip-flops to another XOR gate. It may be noted that the nearest common ancestor of these two leaf nodes is the root. As shown in Fig. 11, the leaf scan flip-flops of branches b and e are connected to the inputs of XOR2. Clearly, an error at root node 1 will be cancelled at the output of XOR2. Since an error therein can be observed through XOR1, this aliasing does not matter. Finally, the gate XOR3 is used to compact the branches $\{c, d\}$.

In order to design the above compactor systematically, we make use of some special properties of the scan tree. From the construction process (Procedure_1), it is evident that the number of nodes

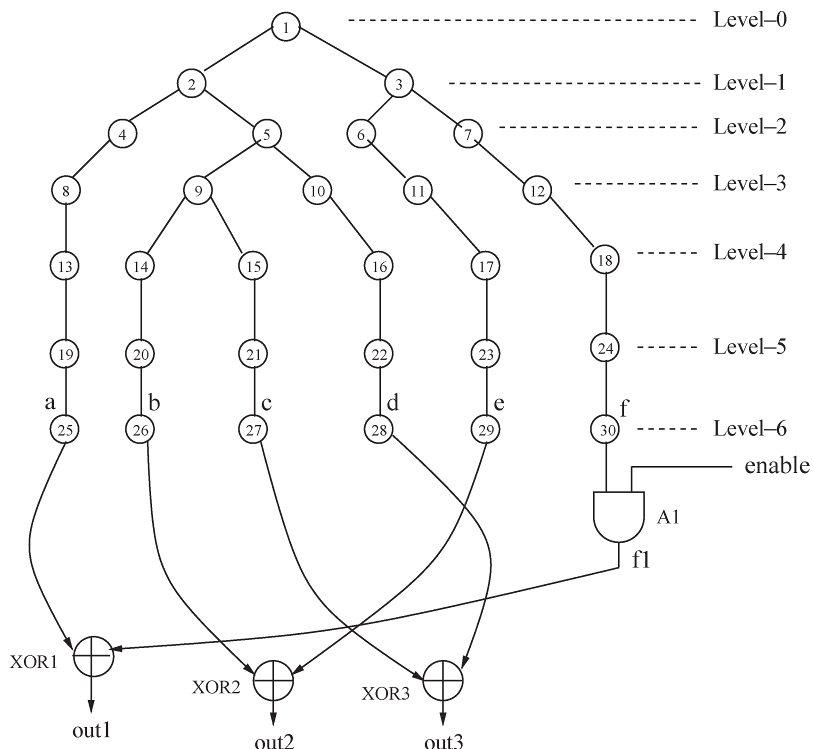

Fig. 11. Proposed compactor circuit for a scan tree.

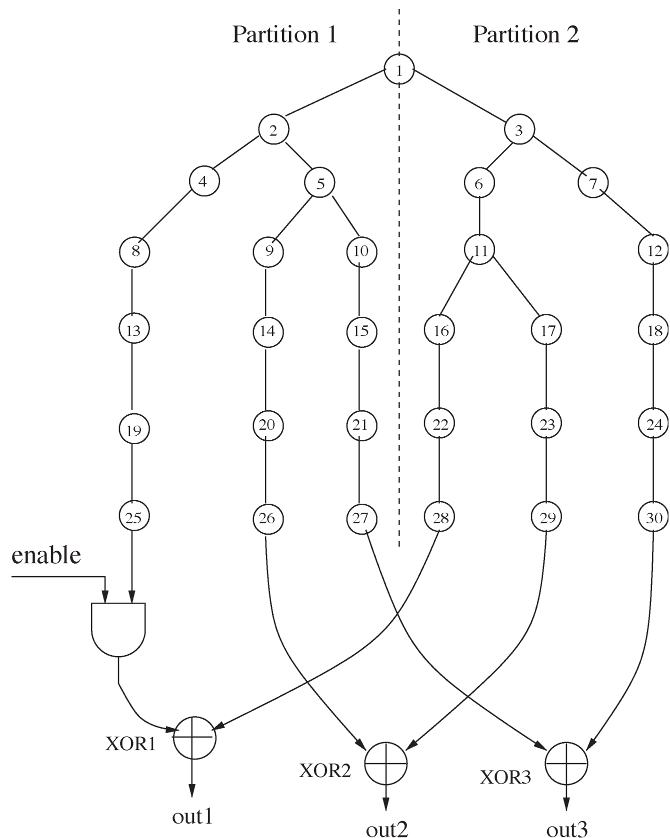

Fig. 12. Equipartitioned scan tree.

(flip-flops) in level l of the tree is equal to or smaller than those in level $l+1$. Further, the tree has equal number of nodes along all the paths. Thus, it is always possible to construct an *equipartitioned tree* as shown in Fig. 12, where in each level following the root, the difference in the number of nodes lying on the two sides of the vertical partition (in dotted lines) is either 0 or 1 and the partition line does not intersect any of the tree edges. The scan tree shown in Fig. 11 can be reconstructed as that in Fig. 12 by slightly modifying Procedure_1, as stated above. In other words, for any pair of leaf nodes, one chosen from the left subset and the other from the right subset of the partition, the only common ancestor will be the root node. This has two implications: 1) the compactor can be easily designed by choosing any two inputs, one from each side of the partition to feed an XOR gate; and 2) at most one AND gate is required to guarantee no aliasing for single errors. The compactor designed in this way will have only one layer of two-input XOR gates, thus achieving nearly 50% space compaction ratio.

To improve the compactor, other sophisticated techniques as in [22]–[24] may be used. It is also shown that the aliasing probability of single stuck-at faults in a CUT at the outputs of the XOR-tree-based compactor is almost zero or negligible [21].

TABLE I
RESULTS ON FULL-SCAN ISCAS'89 CIRCUITS

Circuit name	Number of flip-flops (n_f)	# Test patterns (T_n)	Test cycles	Fault coverage (%)
$s5378$	179	112	20227	100
$s9234$	211	163	34604	100
$s15850$	534	116	62478	99.11
$s38417$	1636	105	173416	99.45
$s38584$	1426	136	195362	98.50

V. EXPERIMENTAL RESULTS

The proposed algorithms are implemented in *C* on a *SUN SPARC ULTRA*-60 (450 MHz) workstation in *SOLARIS* 5.8 environment and are run on several ISCAS'89 benchmark circuits. The test patterns are generated using the TetraMax tool of Synopsys.

TABLE II
RESULTS ON SCAN TREE ARCHITECTURE OBTAINED BY THE PROPOSED ALGORITHM

Circuit name	Depth of the scan tree (n_l)	# Test vectors in ST-mode (T_{st})	Undetectable faults in ST-mode	# Serial test patterns (T_s)	Fault coverage (%)	Total test cycles	Test time saving (%)	CPU time (sec)	Fault coverage (%) [11]	Total test cycles [11]
$s5378$	34	95	153	19	100	6844	66.16	22.4	98.78	4472
$s9234$	37	137	1012	41	100	13968	59.63	67.1	93.16	19391
$s15850$	51	158	185	27	99.11	23061	63.09	160.1	96.29	32604
$s38417$	112	162	231	38	99.45	82060	52.68	225.2	–	–
$s38584$	104	121	302	25	98.50	49764	74.53	211.6	95.23	71226

TABLE III
COMPARISON OF THE PROPOSED SCAN TREE WITH [10]

Circuit name	# Test patterns [10]	Depth of the scan tree [10]	# Bits in scan tree mode [10]	Depth of the scan tree (in our case)	# Bits in scan tree mode (in our case)
$s5378$	100	94	9400	34	3230
$s9234$	111	115	12765	37	5069
$s15850$	97	196	19012	51	8058
$s38417$	87	546	47502	112	18144
$s38584$	114	359	40926	104	12584

TABLE IV
RESULTS ON THE PROPOSED COMPACTOR FOR DIFFERENT BENCHMARK CIRCUITS

Circuit name	# Compacted scan outputs	# Scan outputs as in [10]	Area overhead of the proposed compactor (%)	Area overhead (%) [21]	Area overhead (%) [20]
$s5378$	5	4	0.02	–	13.60
$s9234$	7	7	0.01	–	8.83
$s15850$	8	17	0.01	3.38	12.22
$s38417$	10	52	0.01	6.98	15.20
$s38584$	8	13	0.01	3.47	13.62

The experimental results for the full-scan circuits are presented in Table I. The third and fourth columns report the number of test patterns and the number of cycles necessary to test the CUT, respectively, in full serial scan mode. The last column reports the corresponding fault coverage in full serial scan mode. The test application time (cycles) in full serial scan mode is calculated as $n_f \times (T_n + 1)$, where n_f is the number of flip-flops, and T_n is the number of test patterns. This accounts for only scan-in and scan-out cycles.

The results on scan tree design for ISCAS'89 circuits using the proposed algorithms are shown in Table II. The columns in the table represent the circuit name, depth of the scan tree generated by our algorithm, the number of test patterns in ST mode (T_{st}), the number of faults that cannot be detected under ST mode, the number of serial test vectors (T_s), total fault coverage, the number of cycles necessary to test the circuits, percentage of test time saved compared to full-scan technique, and CPU execution time. The last two columns show the corresponding fault coverage and test cycles needed in an earlier approach [11]. The test application time in the scan tree is computed as $n_l \times (T_{\mathrm{st}} + 1) + n_f \times (T_s + 1)$, where n_l is the depth of the scan tree, T_{st} is the number of test patterns in ST mode, and T_s is the number of test patterns in serial mode. Fault coverage (FC) is defined as

$$\mathrm{FC} = \frac{\text{Faults detected by } (T_{\mathrm{st}} + T_s)}{\text{Total faults present in the circuit}} \times 100\%.$$

The results show that the proposed technique significantly reduces the test application time while achieving high fault coverage, and the fault coverage is the same as that obtained in full serial scan mode.

In Table III, we compare the results on tree depth and the number of test bits required in scan tree mode with those reported by Miyase and Kajihara [10]. The scan tree generated by our algorithm has much smaller depth compared to that in [10] for a compact test set of comparable size. The number of scan-in bits in tree mode also reduces significantly. However, the method in [10] does not use serial mode. Since we aim for achieving high fault coverage, we include serial patterns, and thus, the total test application time becomes larger than that in [10], but it still remains much smaller than that obtained by an earlier hybrid approach [11].

The proposed compactor, an example of which is shown in Fig. 12, has been implemented in 0.18-μm technology (provided by the National Semiconductor, USA) for each of the circuit as listed in Table I. The synthesis of these circuits has been carried out by the Design Analyzer tool of Synopsis. Table IV shows the result for the proposed compactor. The columns in the table present the circuit name, number of compacted outputs, number of scan outputs as in [10], area overhead of the proposed compactor (including the AND gate and the counter), and area overhead reported in [21]. The last column presents the compactor area overhead for scan forest [20]. Thus, the scan-out data volume feeding an MISR is also reduced significantly at the cost of negligible compactor overhead.

VI. CONCLUSION

A new design of scan tree architecture that reduces test application time and test data volume is described in this paper. The method is particularly suitable for a highly compact test set, for which the flip-flops may have very weak or almost no compatibility among

themselves. The technique employs a hybrid approach of applying test vectors in both broadcast and serial mode to generate a scan tree of small depth and to guarantee high fault coverage. The scan tree also leads to a simple design of an XOR tree compactor, which can be used to compact the scan-out data as well. Experimental results show that the method performs favorably compared to earlier schemes. In this approach, we however did not consider the geometrical constraints imposed by physical locations of the flip-flops while constructing the tree and did not address the associated routing problems. Such layout-aware design of the scan tree for reduction of test application time/test data volume may be studied in the future.

Acknowledgment

The authors would like to thank the anonymous reviewers for their suggestions that greatly helped in improving this paper.

References

[1] N. Nicolici and B. M. Al-Hashimi, “Multiple scan chains for power minimization during test application in sequential circuits,” *IEEE Trans. Comput.*, vol. 51, no. 6, pp. 721–734, Jun. 2002.
[2] M. S. Quasemand and S. Gupta, “Designing reconfigurable multiple scan chains for systems-on-chip,” in *Proc. IEEE VLSI Test Symp.*, Apr. 25–29, 2004, pp. 365–371.
[3] S. Y. Lee and K. K. Saluja, “An algorithm to reduce test application time in full scan designs,” in *Proc. Int. Conf. Comput.-Aided Des.*, Nov. 1992, pp. 17–20.
[4] E. M. Rudnick and J. H. Patel, “An genetic approach to test application time reduction for full scan and partial scan circuit,” in *Proc. Int. Conf. VLSI Des.*, Jan. 1995, pp. 288–293.
[5] D. Xiang, M.-J. Chen, J.-G. Sun, and H. Fujiwara, “Improving test effectiveness of scan-based BIST by scan chain partitioning,” *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 24, no. 6, pp. 916–927, Jun. 2005.
[6] Y. Zorian, “Test requirements for embedded core-based systems and IEEE P1500,” in *Proc. Int. Test Conf.*, Nov. 1997, pp. 191–199.
[7] F. F. Hsu, K. M. Butler, and J. H. Patel, “A case study on the implementation of the Illinois Scan Architecture,” in *Proc. Int. Test Conf.*, Oct. 30–Nov. 1, 2001, pp. 538–547.
[8] M. A. Shah and J. H. Patel, “Enhancement of the Illinois Scan Architecture for use with multiple scan inputs,” in *Proc. DATE*, Mar. 4–8, 2002, pp. 368–375.
[9] M. A. Shah and J. H. Patel, “Enhancement of the Illinois Scan Architecture for use with multiple scan inputs,” in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI*, Feb. 19–20, 2004, pp. 167–172.
[10] K. Miyase and S. Kajihara, “Optimal scan tree construction with test vector modification for test compression,” in *Proc. ATS*, Nov. 16–19, 2003, pp. 136–141.
[11] Y. Bonhomme, T. Yoneda, H. Fujiwara, and P. Girard, “An efficient scan tree design for test time reduction,” in *Proc. 9th IEEE ETS*, May 23–26, 2004, pp. 174–179.
[12] H. Yotsuyanagi, T. Kuchii, S. Nishikawa, M. Hashizume, and K. Kinoshita, “Reducing scan shifts using folding scan trees,” in *Proc. 12th ATS*, Nov. 16–19, 2003, pp. 6–11.
[13] H. Yotsuyanagi, T. Kuchii, S. Nishikawa, M. Hashizume, and K. Kinoshita, “On configuring scan trees to reduce scan shifts based on a circuit structure,” in *Proc. IEEE Int. Workshop Electron. Des., Test and Appl.*, Jan. 28–30, 2004, pp. 269–274.
[14] K. Miyase, S. Kajihara, and S. M. Reddy, “Multiple scan tree design with test vector modification,” in *Proc. Asian Test Symp.*, Nov. 15–17, 2004, pp. 76–81.
[15] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York: Freeman, 1979.
[16] D. Brelaz, “New methods to color the vertices of a graph,” *Commun. ACM*, vol. 22, no. 4, pp. 251–256, Apr. 1979.
[17] C. Fleurent and J. A. Ferland, “Genetic and hybrid algorithms for graph coloring,” *Ann. Oper. Res.*, vol. 63, no. 3, pp. 437–461, Jun. 1996.
[18] S. Zhang, S. C. Seth, and B. B. Bhattacharya, “On finding consecutive test vectors in a random sequence for energy-aware BIST design,” in *Proc. 18th Int. Conf. VLSI Des.*, Jan. 3–7, 2005, pp. 491–496.
[19] H. Cui, S. C. Seth, and S. K. Mehta, “Modeling fault coverage of random test patterns,” *JETTA*, vol. 19, no. 3, pp. 271–284, Jun. 2003.
[20] D. Xiang, S. Gu, J.-G. Sun, and Y. L. Wu, “A cost-effective scan architecture for scan testing with nonscan test power and test application cost,” in *Proc. Des. Autom. Conf.*, Jun. 2–6, 2003, pp. 744–747.
[21] D. Xiang, K. Wei Li, and H. Fujiwara, “Design for cost effective scan testing by reconfiguring scan flip-flops,” in *Proc. Asian Test Symp.*, Dec. 18–21, 2005, pp. 318–323.
[22] P. Wohl, J. A. Waicukauski, and T. W. Williams, “Design of compactors for signature analyzers in built-in self-test,” in *Proc. Int. Test Conf.*, Oct. 30–Nov. 1, 2001, pp. 54–63.
[23] S. Mitra and K. S. Kim, “X-compact: An efficient response compaction technique for test cost reduction,” in *Proc. Int. Test Conf.*, Oct. 7–10, 2002, pp. 311–320.
[24] S. Mitra and K. S. Kim, “X-compact: An efficient response compaction technique,” *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 23, no. 3, pp. 421–432, Mar. 2004.

Efficient Design for Testability Solution Based on Unsatisfiability for Register-Transfer Level Circuits

Loganathan Lingappan and Niraj K. Jha

***Abstract*—In this paper, we present a novel and accurate method for identifying design for testability (DFT) solutions for register-transfer level (RTL) circuits. Test generation proceeds by abstracting the circuit components using input/output propagation rules so that any justification/propagation event can be captured as a Boolean implication. Consequently, the RTL test generation problem is reduced to a satisfiability (SAT) instance. If a given SAT instance is not satisfiable, then we identify Boolean implications (also known as the unsatisfiable segment) that are responsible for unsatisfiability. We show that adding DFT elements is equivalent to modifying these clauses such that the unsatisfiable segment becomes satisfiable. The proposed DFT technique is both fast and accurate as it is applicable to RTL and mixed gate-level/RTL circuits and uses exact unsatisfiability conditions to identify the DFT solutions.**

***Index Terms*—Design for testability, register-transfer level, satisfiability, test generation.**

I. Introduction

The complexity of automatic test-pattern generation (ATPG) for large sequential circuits has forced designers to look for solutions that reduce test generation time and increase fault coverage for a given sequential circuit. Such solutions belong to the realm of design for testability (DFT). In this paper, we propose one such DFT technique based on unsatisfiability. The proposed technique is applicable to register-transfer level (RTL) and mixed gate-level/RTL designs and is very fast. In addition, it is very accurate in its ability to pinpoint exactly the causes behind untestability.

A. Related Work

Popular DFT solutions that are widely used in the industry are full-scan design, partial-scan design, and test point insertion [1]. In full-scan design, each flip-flop in the given sequential circuit belongs to

Manuscript received August 16, 2005; revised May 27, 2006. This work was supported by SRC under Contract 2002-TJ-1039. This paper was recommended by Associate Editor K. Chakrabarty.

The authors are with the Department of Electrical Engineering, Princeton University, Princeton, NJ 08544 USA (e-mail: llingapp@princeton.edu; jha@princeton.edu).

Digital Object Identifier 10.1109/TCAD.2006.888268

0278-0070/$25.00 © 2007 IEEE