Diagnosis of Multiple Scan-Chain Faults in the Presence of System Logic Defects

Zhen Chen
Tsinghua National Laboratory for Information Science and Technology

Sharad C. Seth
University of Nebraska-Lincoln, seth@cse.unl.edu

Dong Xiang
Tsinghua University, Beijing

Bhargab B. Bhattacharya
Indian Statistical Institute, Kolkata, India, bhargab@isical.ac.in

Diagnosis of Multiple Scan-Chain Faults in the Presence of System Logic Defects

Zhen Chen[1], **Sharad Seth**[2], **Dong Xiang**[3], **Bhargab B. Bhattacharya**[4]
[1,3]Tsinghua National Laboratory for Information Science and Technology
[1]Dept. of Comp. Sci. and Techn., [3]School of Software, Tsinghua University, Beijing 100084, China
[2]Computer Sci. and Eng., University of Nebraska-Lincoln, Lincoln NE 68588-0115, U.S.A.
[4]Indian Statistical Institute, Kolkata, India

Abstract —We present a combined hardware-software based approach to scan-chain diagnosis, when the outcome of a test may be affected by system faults occurring in the logic outside of the scan chain. For the hardware component we adopt the double-tree scan (DTS) chain architecture, which has previously been shown to be effective in reducing power, volume, and application time of tests for stuck-at and delay faults. We develop a version of flush test which can resolve a multiple fault in a DTS chain to a small number of suspect candidates. Further resolution to a unique multiple fault is enabled by the software component comprising of fault simulation and analysis of the response of the circuit to test patterns produced by ATPG. Experimental results on benchmark circuits show that near-perfect scan-chain diagnosis for multiple faults is possible even when a large number of random system faults are injected in the circuit.

Keywords: Double tree scan, Scan chain diagnosis, system logic defects.

I. Introduction

As the scan chains provide the critical test infrastructure through which all test data must pass, establishing the integrity of scan chains is essential for drawing meaningful conclusions from any scan-based test. In this paper, we focus on diagnosis of multiple faults in scan chains. This diagnosis step could be followed by another step in which faulty nodes in the system logic are identified with the knowledge of the scan-cell faults.

Our scan-chain diagnosis process starts with the application of a flush test (aka chain pattern) to detect scan cell faults. The flush test can only partially resolve the faults and fault types [1], therefore additional steps that analyze failure information of scan patterns are required to complete the diagnosis. Many early scan chain diagnosis methods make the unrealistic assumption that the system logic is fault-free [14]. The diagnosis process is considerably more challenging for a compound fault model in which system logic faults can co-occur with scan chain faults and can corrupt the test results obtained in the absence of system logic faults [2, 3, 6, 16]. Prior studies that allow compound faults in scan chain diagnosis, however, are limited to solutions that solve specific cases or can tolerate only a small number of logic defects for perfect diagnosis.

A recent survey provides a convenient classification of the many schemes that have been proposed for scan chain diagnosis and discusses their pros and cons [1]. In the *tester-based* approach, the tester works in conjunction with a physical failure analysis (PFA) device to scan-shift test patterns, capture responses, and analyze defective responses at different locations to identify failing scan cells [8, 9]. The *hardware-based* schemes employ special scan-chain or scan-cell designs to facilitate diagnosis [10, 11]. *Software-based* methods algorithmically analyze the test data for diagnosis. These can be further broken down into two broad sub-categories: (1) *simulation based* in which available test patterns are fault simulated and analyzed [2, 4, 12], and (2) *deterministic diagnostic pattern generation (DDPG) based* which target individual scan cells on a faulty scan chain for test generation [13, 15].

In this paper, we propose a hybrid hardware-software based approach. For the hardware scheme we adopt the double-tree scan (DTS) which duplicates a scan chain in function and provides a unified approach to reducing the power, volume, and application time of tests for stuck-at and delay faults [7]. We show that the flush test can already narrow the ambiguity about each component of a multiple fault to at most three faults in a scan chain. The software scheme compares the simulation results of available test patterns against observed responses and resolves the ambiguity of each fault component down to a unique fault in the presence of hundreds of random sys- tem logic defects in the circuit. To the best of our knowledge, this is the first method that can achieve near-perfect resolu- tion in the presence of hundreds of system logic defects

The rest of the paper is organized as follows. Section II provides the background on the DTS architecture necessary for an understanding of this paper. It also precisely defines the fault model for scan-chain faults used in the paper. In Section III, a flush test for a DTS scan chain is developed. Software based techniques for further resolving the fault to a single scan cell are discussed in Section IV. The experimental set up and results are discussed in Section V. The paper concludes in Section VI with a discussion of possible extensions of this work.

II. Background

In the DTS approach, each linear scan chain is replaced by its equivalent nonlinear DTS chain. In this section we cover the salient aspects of the double-tree scan (DTS) architecture that are essential to an understanding of this paper. We also define the fault model used for scan-chain and system logic faults.

A. Double-Tree Scan

Structure: A complete binary tree has k levels (with the root assumed at level 0) and consists of 2^k leaf nodes and (2^k-1) internal nodes. The DTS structure resembles two complete k-level binary trees whose leaf nodes are merged pair-wise. Thus, a full double-tree DTS(k) consists of $N = (2^k-1+2^k+2^k-1) = 3*2^k-2$ nodes. Each node of the tree represents a scan flip-flop. The edges in the tree are directed from top to bottom. A directed edge (i, j) in the DTS indicates that Q(i), the Q-output of the flip-flop i, drives D(j), the D-input of the flip-flop j. For each node with in-degree 2 (i.e., a merge node) in the bottom half of the DTS, a 2-1 MUX is needed to select the predecessor flip-flop during the scan-load operation. Fig. 1 shows a full DTS(2) with 10 nodes. A DTS with an arbitrary number of nodes can be obtained by either pruning a larger full DTS or chaining together multiple DTS structures of smaller size [7].

The full DTS structure can be partitioned into three roughly equal parts according to the type of nodes in the tree. The top

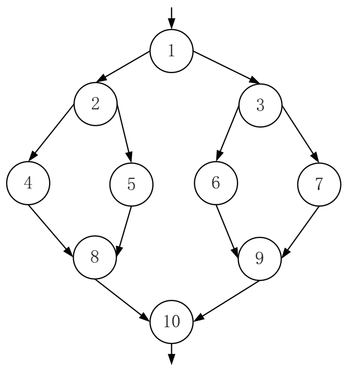

Figure 1: A DTS chain with 10 nodes.

part consists of 2^k-1 forking nodes with in-degree 1 and out-degree 2; the middle part consists of just one level of 2^k nodes, with in-degree and out-degree equal to 1; and the bottom part consists of 2^k -1 merging nodes with in-degree 2 and out-degree 1.

Operation: A defining characteristic of DTS is that it can functionally replace a scan chain, by connecting the topmost (source) node to the scan-in signal and the bottommost (sink) node to the scan-out signal. The full DTS(k) has 2^k overlapping scan paths, each of length $(2k + 1)$, from the source to the sink. The control unit to load and unload the DTS repeatedly selects a source-to-sink scan-shift path in each clock cycle, so that externally the DTS appears indistinguishable from a scan chain of length N, where N is the number of FFs in the DTS. However, in each clock cycle, only O(log N) FFs are enabled, thus providing O($N/logN$) improvement in test power per cycle. This translates to a reduction by a factor of over 250 for a DTS chain of 6000 FFs. By using a gated test clock and rippling it from the bottom of the tree with the data input still at the top of the tree, not only is the race condition between the data and the control signals completely eliminated but significant reduction in clock power also becomes possible [17].

The original DTS architecture [18] has been shown to be a unified way of reducing power, volume, and application time of tests for stuck-at and delay faults [7].

Implementation: The impact of the DTS architecture on the area, routing, and test power was recently studied for several benchmarks (including a RISC CPU design and several ITC 99 benchmarks) mapped to the IBM 0.18-micron process [17]. The results show a centralized DTS area overhead of 15% over standard scan for most circuits. In all cases, the design area did not have to be artificially expanded to allow more routing resources, i.e., it was possible to use the same standard cell utilization (over 95%) for all versions of the design. Further, for a fabricated design of a sinc decimation filter with 188 functional flip-flops, the measured power savings on random test patterns is 9x-10x, which is consistent with the O($N/logN$) reduction predicted by analysis.

The huge savings in test power provide a means of reducing the are overhead of DTS: by replacing the flip-flop at each node by a chain of N flip-flops, the area overhead of DTS per node is reduced by a factor of N, at the expense of an N times increase in the power consumption. Thus, the measured 18% area overhead over scan of a benchmark with over 6000 flip-flops could be reduced to just 2% by chaining 9 flip-flops at each node, while still realizing a power reduction of over 25. The reduction in area occurs because the control overhead for node implementation remains essentially unchanged.

B. Fault model

Although intermittent faults have been considered [5], most studies on scan-chain diagnosis assume single permanent faults that may be restricted to only stuck-at faults at the cell output, or extended to include timing faults also. With the current density of devices on a chip, a single fault model is hard to justify, therefore, in this paper we allow multiple faults to occur in a DTS chain. Further, the type of components of a multiple faults is also extended to account for the increased complexity of a scan cell in a DTS vs. a linear chain. Our objective is to diagnose a multiple scan chain fault in the presence of system logic faults, where the latter can be quite large (up to 50 in our experiments) number.

In describing the effect of a scan-cell fault during the loading/unloading process, it is helpful to define the upstream and downstream relations between two scan cells a and b on a scan path. Cell a is upstream of cell b if a appears before b on the scan path, otherwise, it is downstream. As an example, if cell a is stuck-at-0, all the cells downstream of a will have the value 0 after the loading process. Similarly, the response values for all the cells passing through a during the unloading process would appear to have the value 0 after the unloading process. It is noteworthy that, unlike a linear scan chain, these relations do not hold for all pairs of cells in a DTS chain (e.g. cells 4 and 6 in Fig 1).

The types of fault allowed at a DTS node are related to either its storage or switching function. For the storage function we permit the standard stuck-at and timing faults to occur at a scan cell, i.e. the cell output could be SA0, SA1, STR (slow-to-rise), and STF (slow-to-fall). For a failure in the switching function, we assume a clock gating scheme, similar to that described in [17], where the test clock and test data propagate from the opposite ends of the double-tree scan. Specifically, the test clock is routed bottom-to-top, via demux switches in the bottom part and mux switches in the top part, while the data terminals of the scan cells are directly connected top-to-bottom, according to the DTS topology. We assume that the switching failures are related to the select functions of the routing switches. Specifically, a switch could be stuck-right or stuck-left depending on whether the switch always passes to the right or left due to the failure of the select function.

To illustrate how switch failures affect the operation of a DTS chain, consider the fault: switch (demux) at node 10 stuck-right in Fig. 1. As a result of this fault, any time a scan path in the left half of the double-tree scan is normally selected by clock gating, a scan path in the right half would be selected instead. For example, instead of the path (1, 2, 5, 8, 10), the path (1, 3, 7, 9, 10) will be selected. Equivalently, we can denote this fault as either missing-edge (1,2) or missing-edge (8,10), from the perspective of the DTS operation.

A switch failure in the top part can be illustrated by considering the fault: switch (mux) at node 2 stuck right in Fig. 1. As a result, the test clock arriving at node 4, from nodes 10 and 8, would be blocked from further propagation to nodes 2 and 1. Thus, whenever, the path (1,2,4,8,10) is selected for shifting in the normal operation, nodes 1 and 2 will hold their values and the shift operation would occur only on the sub-path (4,8,10). Equivalently, we can denote this fault as a missing-clock fault on nodes 1 and 2.

In summary, our fault model includes multiple faults with components of the following types: Storage Faults: stuck-at-0 (SA0), stuck-at-1 (SA1), slow-to-rise (STR), and slow-to-fall (STF). Switching Faults: missing-edge and missing-clock.

Further, for diagnosis, we only consider multiple faults in which no masking relationship exists between fault components. For example, in Figure 2, instead of the multiple fault

Table 1: Loading and unloading process for a flush pattern

Cycle	Active scan path	Node index									
		1	2	3	4	5	6	7	8	9	10
1	1-2-5-8-10	b1									
2	1-3-7-9-10	b2		b1							
3	1-2-4-8-10	b3	b2	b1							
4	1-3-6-9-10	b4	b2	b3			b1				
5	1-2-5-8-10	b5	b4	b3		b2	b1				
6	1-3-7-9-10	b6	b4	b5		b2	b1	b3			
7	1-2-4-8-10	b7	b6	b5	b4	b2	b1	b3			
8	1-3-6-9-10	b8	b6	b7	b4	b2	b5	b3		b1	
9	1-2-5-8-10	b9	b8	b7	b4	b6	b5	b3	b2	b1	
10	1-3-7-9-10	b10	b8	b9	b4	b6	b5	b7	b2	b3	b1 (r1)
11	1-2-4-8-10	b11	b10	b9	b8	b6	b5	b7	b4	b3	b2 (r2)
12	1-3-6-9-10	b12	b10	b11	b8	b6	b9	b7	b4	b5	b3 (r3)
13	1-2-5-8-10		b12	b11	b8	b10	b9	b7	b6	b5	b4 (r4)
14	1-3-7-9-10		b12		b8	b10	b9	b11	b6	b7	b5 (r5)
15	1-2-4-8-10				b12	b10	b9	b11	b8	b7	b6 (r6)
16	1-3-6-9-10				b12	b10		b11	b8	b9	b7 (r7)
17	1-2-5-8-10				b12			b11	b10	b9	b8 (r8)
18	1-3-7-9-10				b12				b10	b11	b9 (r9)
19	1-2-4-8-10								b12	b11	b10 (r10)
20	1-3-6-9-10								b12		b11 (r11)
21	1-2-5-8-10										b12 (r12)

(node-2 SA0, node-4 SA0), we will consider the single fault node-2 SA0, because it masks the effect of the other fault. Similarly, the missing edge fault on edge (8,10) masks the missing edge fault on edge (4,8).

III. FLUSH TESTS FOR DTS

In this section, first, we discuss a defining feature of DTS that helps improve its fault resolution significantly, as compared to the linear scan chain. Then, we develop flush tests for stuck-at and timing faults. For ease of exposition, we first discuss the diagnosis of single faults, then extend the same to multiple faults in Section V.

A. Fault Resolvability of DTS for Flush Tests

As noted in the last section, among the many source-to-sink scan paths in the DTS only one is active during any clock cycle. We can use this property to obtain high diagnostic resolution. We assume that, a flush pattern denoted by (b1, b2, . . . , b10) is loaded into the DTS chain in Fig. 1 while the observed response, after unloading, is denoted by (r1, r2 . . . , r10). The loading and unloading processes are shown in Table 1. If we index the scan paths from left to right as P1, P2, P3, P4, the loading sequence shown in the table cycles through the paths in the following order: (P2, P4, P1, P3). In the flush tests, we use this loading sequence.

Each bit scanned into the tree passes through exactly one of these four paths before being observed, e.g. bit b1 passes through path P3 consisting of scan cells (1, 3, 6, 9, 10) and then is observed as r1. If r1 differs from b1, we can conclude that the faulty scan cell must be one of the five cells in P3. In general, each failure bit can be ascribed precisely to one scan path. Further, after analyzing the paths for all the failure bits, we can narrow down the suspect list of faulty cells to one or two candidates. This result can be demonstrated by means of an example. If only scan-path P1 has a fault (P2, P3, and P4 are fault-free), the faulty scan cell must be 4 because every other cell on P1 appears also on at least one other path. If failures are observed for both P1 and P2 then the faulty cell must be either 2 or 8, because these are the only two cells that are common to both P1 and P2. On the other hand, if failures are observed on all four paths, the faulty cell must be 1 or 10 by the same logic.

In contrast, for a linear scan chain, the response analysis of a flush test can only narrow the suspect list to a chain segment bounded to upper-bound (UB) and lower-bound (LB) cells and, in the worst case, the suspect list might correspond to all the cells in the scan chain.

B. Flush test for non-timing faults

In this subsection, we consider the stuck-at, missing-edge, and missing-clock faults. First, we give the analysis on how these faults affect the loading and unloading process. Next, we provide a flush test for detecting these faults.

From the previous discussion, we know that a SA-0(1) fault causes all the bits passing through the faulty node to be observed as 0(1). For example, if there is a SA-0 fault for node 2 in Fig. 1, all the bits passing through P1 and P2 will be observed as 0, no matter what kind of flush pattern is loaded.

We can also illustrate the effect of a missing-edge fault by an example. Suppose in Fig. 1, there is a missing-left-edge fault at node 1, i.e., the clock at node 1 always routes to the right so that instead of the normal cycle of activated paths (P2, P4, P1, P3), we would get (P4, P4, P3, P3). In other words, the paths P1 and P3 would never be activated and the values of nodes (2, 4, 5, 8) will never get out. Consequently, the fault results in a pruned DTS in operation, and we cannot load as many values as the number of nodes in the DTS. We can get the number of inaccessible nodes quite simply by initially loading the tree with all 0s and following it will all 1s. The number of 0s that come out during the second pattern load would represent the number of inaccessible nodes.

The missing-clock fault means that the faulty node and all its ancestors to the source node are never clocked, but all the downstream nodes on the selected path will be clocked. For example, in Fig. 1, if the fault blocks the propagation of clock from node 2 to node 1, whenever path P1 (or P2) is selected for shifting, node 1 would hold its old value because of the missing clock. Further, the shift operation would proceed normally for the remaining path 2-4-8-10 (or 2-5-8-10). If the value at node 2 before the flush test is x, this missing-clock fault would appear as equivalent to the stuck-at-x fault at node 2. This is because node 2 is no longer controllable from the input and its value x is propagated to all the downstream nodes. If x=0, it is equivalent to the SA-0 fault otherwise, it is equivalent to the SA-1 fault.

Next, we give a flush test for diagnosing single non-timing faults:

- Pattern 1: Apply all 0s.
- Pattern 2: Apply all 1s and observe the output.
- Pattern 3: Apply all 1s.
- Pattern 4: Apply all 0s and observe the output.

Note that because the four test patterns involve no signal transitions, these tests would mask any timing faults. The expected response for for the second pattern (R2) is all 0s and for the fourth pattern (R4), it is all 1s. Any deviation from these responses signals the occurrence of a non-timing fault. Further, by analyzing the deviation more closely, we can distinguish the missing-edge faults from the stuck-at and missing-clock faults and narrow the suspect list to a small number of candidates. This can be explained as follows. For a missing-edge fault, as explained earlier, R2 will include some 1s, equal in number to the inaccessible nodes, conversely, R4 will include an equal number of 0s. For example, if R2 has four 1s and R4 has four 0s, we can conclude that a DTS sub-tree of size 4 is made inaccessible, hence the missing-edge fault can be attributed to node 1, but we cannot say whether the left or the right edge is missing at this node, i.e. whether nodes (2,4,5,8) or nodes (3,6,7,9)are made inaccessible.

For a stuck-at or missing-clock fault, errors will appear in only one of the two response vectors, hence these type of faults are easily distinguished from the missing-edge faults. For example, if R2 is error-free but we observe errors (0s) for paths P1 and P2 in R4, then we can conclude that node 2 or 8 may have a SA0 fault.

A missing-clock fault and a SA-fault have different effects during the scan process, since the upstream nodes of the faulty node with missing-clock fault cannot load values into the faulty node. However, a missing-clock fault will have the same syndrome as SA-0 or SA-1 for the flush test, hence we cannot distinguish it from the stuck-at faults by flush tests.

In summary, the flush test narrows the suspect list to one or two type of faults and to a small number of actual faults within each type. For a further refinement of the suspect list, we will need to use scan-based patterns, as described in Section IV.

C. Flush test for Timing Faults

For identifying the faulty cell with a timing fault, a flush test should meet two goals: (1) Ensure that both transitions ($0 \rightarrow 1$ and $1 \rightarrow 0$) pass through each cell. (2) Generate different syndromes for different faulty scan cells in order to achieve a high diagnostic resolution. The linear scan chain cannot achieve high diagnostic resolution for timing faults by flush tests, since the syndromes for all scan cells are the same. Here, we make use of the flexibility of DTS to improve the resolution.

We show that the flush patterns 001001001001 and 110110110110, for DTS(2) in Fig. 1, meet the two goals (for DTS(k), these flush patterns correspond to 2^k consecutive 001 and 110). Although there are 10 nodes in Fig. 1, the flush pattern needs 12 bits so as to ensure that at least two transitions passing through each scan cell. Scan cell 4 in Fig. 1 will have only one transition (from b4 to b8) passing through it, if only the first 10 bits are used in the flush pattern.

We first show that the flush patterns can detect any faulty scan cell in the DTS chain. Next, we show that the resolution of syndromes for different faulty scan cells is less than or equal to 2.

Table 2: Fault syndromes for a single STR fault

Faulty cell	Failure bits in observed response
1	r3 , r6 , r9 , r12
2	r6 , r12
3	r3 , r9
4	r12
5	r6
6	r9
7	r3
8	r6 , r12
9	r3 , r9
10	r3 , r6 , r9 , r12

From a simulation of the two flush patterns, it can be verified that both the rising transition and falling transition pass through each scan cell. For example, for the flush pattern is (b1, b2, . . . , b12) = 001001001001, two $0 \rightarrow 1$ transitions pass through scan cell 2 in cycles 7 and 13 and there is a $1 \rightarrow 0$ transition for scan cell 2 in cycle 9 . Therefore, any timing fault in a cell will also result in a failure of the flush test.

The syndromes for different faulty scan cells are shown in Table 2, given that the flush pattern is 001001001001 and fault is slow-to-rise (STR). For example, from Table 1, we see that test bits b4, b8 and b12 pass the scan cell 4, and there is a rising transition in clock cycle 15 when the value changes from b8 to b12. If scan cell 4 has a STR fault, the value of b12 is corrupted by the fault, and the syndrome is that r12 is different from the expected value b12.

From the example, we can see that only the symmetric scan cells (e.g. cells 2 and 8 or cells 3 and 9 in Fig. 1) have the same syndrome when they have faults. Therefore, we can narrow the suspected scan cells to 1 or 2 by observing their unloading response.

IV. REFINING FAULT RESOLUTION

In this section, we show how the fault resolution can be further improved by additional tests. The strategy we use for timing faults differs from that used for no-timing faults. For simplicity, we first talk about the single fault model, then give the multiple fault scheme based on it in Section V.

A. Non-timing Faults

After the flush test, a SA-fault can be narrowed down to two symmetric nodes (one in the top part and the other in the bottom part) or one node (in the middle part); a missing-edge fault can be narrowed down to some symmetric parts (for example part (2,4,5,8) or (3,6,7,9) in Fig.1); a missing-clock fault can be narrowed down to one candidate, but we cannot distinguish it from the SA-fault in the same position. Therefore, we first give the scheme to distinguish two SA-faults in the symmetric nodes and the missing-clock fault in the top part. Next, we attempt to identify the inaccessible nodes for missing-edge faults.

Our strategy for disambiguating stuck-at faults and missing-clock faults is based on the failure information for production test patterns. In order to describe the idea, we use the following notation.

Assume that the two symmetric suspect cells in the DTS chain are a and b, where b is downstream of a. Then, the set of all scan cells downstream of a and upstream of b, exclusive of a and b, is denoted by (a, b). If cell b is included, we denote the set as (a, b]. The faults we attempt to distinguish are SA-fault of a, missing-clock fault of a and SA-fault of b. The differences between SA-fault of a and missing-clock fault of a are that the latter may be SA1 or SA0 for different test patterns. During the capture cycle, if node a captures value 1, it will be a SA1 fault. Otherwise, it is a SA0 fault. If for many independently generated test patterns, consistently, only SA1 or only SA0 value is captured for a suspect node, we declare the fault type to be stuck-type, otherwise, it must be a missing-clock fault.

Let $r(a)$ represent the expected observed value for scan cell a and $r^*(a)$ be its real observed value. The observed value could differ from $r(a)$ not only because the applied (loaded) test stimuli are different from their nominal values due to a scan-chain fault but also because of system faults. As these two causes are not easily separated, we can only say that:

$$r^*(a) = \begin{cases} v & \text{if cell } a \text{ is upstream of the SA-v faulty scan cell,} \\ r(a) & \text{if cell } a \text{ is downstream of the faulty scan cell and is unaffected by the system logic defects,} \\ \overline{r(a)} & \text{otherwise.} \end{cases} \quad (1)$$

For each test vector, the response can be classified into two cases:

(1) For at least one cell, say x, in the segment (a, b], the following condition holds: $r^*(x) = \overline{v}$, given that a SA-v fault has occurred in either cell a or cell b. In this case, the fault must be in scan cell a, because $r^*(x) = v$ for all cells x in (a, b] if scan cell b is faulty.

(2) Otherwise, that is, $r^*(x) = v$ for all cells x in the segment (a, b].

If case (1) occurs for any test vector, the diagnosis process terminates, as we can conclude that scan cell a is faulty. If case (2) occurs for all the test vectors, we cannot distinguish the faulty cell: the observed response could be because either scan cell b is faulty or scan cell a is faulty along with system logic faults.

For resolving the ambiguity in the second case, we use a probability-based method that compares the observed response of a test vector to two simulated responses, obtained by assuming the stuck-at fault in cell a and cell b, respectively.

Instead of comparing all the bits in the test response, we only compare them on a subset of bits in the response. We find the set of good scan cells that are not affected in the loading and unloading process, denoted by S_{good}.

For each test vector, we define the following:

- $R_{observed}$: the response observed by ATE. It is the response of fault simulation when both scan chain fault and system logic faults are randomly injected.
- R_a: the response of fault simulation when only scan cell a is faulty and there is no system logic defect.
- R_b: the response of fault simulation when only scan cell b is faulty and there is no system logic defect.

$$score(a) = |\{x | x \in S_{good} \cap R_{observed}(x) = R_a(x)\}|/DFF$$
$$score(b) = |\{x | x \in S_{good} \cap R_{observed}(x) = R_b(x)\}|/DFF$$

Here, DFF is the number of scan cells in the circuit. We claim that scan cell a or b is faulty if $(|\sum score(a) - \sum score(b)| > margin)$ after summing the scores of all the test vectors (the larger one will be the candidate). Otherwise, both scan cell a and b are reported. Here, the parameter $margin$ is introduced to avoid a false decision. We set the value for $margin$ based on the statistics of samples from all the circuits.

For the missing-edge fault, the inaccessible nodes can never be activated in the scan process, thus, their captured values will never get out during the scan process. From the flush test, we can infer the number of inaccessible nodes in the DTS. Take Fig. 1 as an example, if nodes (2,4,5,8) or (3,6,7,9) are inaccessible, only the first 6 bits during the unloading process for a pattern are captured response bits, while the other four bits are error bits. Based on the values of the 6 observed response bits, we can determine which part (nodes (2,4,5,8) or (3,6,7,9)) is inaccessible. Therefore, for a DTS with inaccessible candidate parts, P_1 and P_2, we first calculate the expected response of P_1 and P_2 for each pattern. Then, we compare the expected response of P_1 and P_2 with the observed response in the ATE. Finally, the part whose expected response more matches the observed response is more likely to be the suspect.

B. Timing Faults

Timing faults can be uniquely identified by using another flush pattern, as demonstrated below for the running example.

Assume that the suspected faulty scan cells are 2 and 8, and the timing fault is STR (01→00). In the new flush pattern, we load (0, 0, 1, 1) into scan cells (8, 4, 5, 2) and arbitrary values into other scan cells.

Suppose the four values are scanned into the scan cells by activating paths P1 and P2 in each clock cycle as follows:

- Cycle 1: (P1 is active) 0 into cell 2.
- Cycle 2: (P2 is active) 0 into cell 5 and 0 into cell 2.
- Cycle 3: (P1 is active) 0 into scan cell 4 and 1 into scan cell 2.
- Cycle 4: (P2 is active) 0 into cell 8, 1 into cell 5, and 1 into scan cell 2.

After the loading process, the values in scan cells (8,4,5,2) are (0,0,1,1). For unloading, we activate only path P1. This scheme allows distinguishing the timing fault in cell 2 vs. cell 8 as follows:

If cell 2 is faulty, the value in scan cell 5 after loading will be 0, since the 0→1 transition in cell 2 in Cycle 3 will change to 0→0, and the faulty value 0 is loaded into cell 5 in Cycle 4. However, the loaded values in cells 2, 4, and 8 remain unchanged, independent of whether cell 2 is faulty or fault-free. In the unloading process, we only activate path P1 for 5 consecutive cycles. The values 1 and 0, respectively in cells 4 and 2, define a rising transition that must pass through cell 8. If the rising transition is observed, it indicates that cell 8 is fault-free; hence the timing fault must be in cell 2. Otherwise, scan cell 8 is faulty.

The overhead for this method is a minor enhancement to the DTS control that allows for activating a single scan path for multiple cycles during unloading.

V. Multiple faults diagnosis

For multiple faults, we adopt the law of parsimony, i.e., we attempt to use the smallest number of suspects to explain the faulty syndrome. For example, if stuck-at faults appear to occur on both P1 and P2, the culprits may be nodes 4 and 5 together, or node 2 alone. According to parsimony, we declare node 2 to be the cause.

If there are multiple stuck-at and missing-clock faults, we attempt to find the smallest number of suspects by the flush tests. Then, use scan-based patterns to determine the fault type and location. For example, if we observe SA-0 faults for P1, P2 and P3 in the flush test, the suspected list of nodes will be 2, 8 and 6. Further, we must distinguish between the SA-0 fault at node 2, SA-0 fault at node 8, missing-clock fault at node 2, SA-0 fault at node 6 and missing-clock fault at node 6. We use multiple procedures for single fault model to solve this. For the SA-0 faults at nodes 2 and 8, and missing edge fault of node 2, we use the method in Section IV.A to distinguish

Table 3: Performance of the proposed method

Circuit		1 chain fault			2 chain fault			3 chain fault		
Logic faults		0	10	50	0	10	50	0	10	50
s13207	@1	100	99	98	97	97	94	93	91	86
	@2	100	100	100	99	98	97	95	92	90
	@5	100	100	100	100	100	100	100	99	97
s15850	@1	100	99	97	97	94	93	93	90	87
	@2	100	100	100	99	99	98	96	92	92
	@5	100	100	100	100	100	100	100	100	99
s38417	@1	100	100	100	98	98	94	94	93	89
	@2	100	100	100	100	100	97	99	96	94
	@5	100	100	100	100	100	100	100	98	98
s38584	@1	100	100	99	97	96	93	92	90	85
	@2	100	100	100	100	99	94	96	96	93
	@5	100	100	100	100	100	100	100	98	98
s35932	@1	100	100	100	100	99	97	95	91	90
	@2	100	100	100	100	100	100	98	98	96
	@5	100	100	100	100	100	100	100	100	99

them. For the SA-0 fault at node 6 and missing-clock fault at node 6, we also use the method in Section IV.A to distinguish them. Therefore, we use the method for single fault multiple times to distinguish the multiple faults.

If there are multiple stuck-at, missing-clock faults and missing-edge faults, we first use the method in the last paragraph of Section IV.A to find the locations for missing-edge faults, then, use the probability-based method in Section IV.A to distinguish the stuck-at, and missing-clock faults.

VI. EXPERIMENTAL SET UP AND RESULTS

The proposed method was implemented to demonstrate its efficacy in diagnosing scan chain faults. In the results reported in Table 3, we demonstrate that the proposed method can identify all kinds of faults in the scan chain for almost all the cases.

After the flush tests, we use the method described in Section IV and V to further improve the resolution. For each circuit, we randomly inject 1, 2, or 3 chain faults in a DTS, and report the results for three cases, corresponding to the injection of 0, 10, or 50 random logic defects, respectively, in the non-scan part of the circuit. For each case, we carried out 100 trials. In order to capture non-traditional fault modes, a flipping line fault model is used for the logic defects, i.e., we assume that the defect complements the normal logic value that would occur on that line. Thus, with 10 injected faults, 10 randomly selected lines in the circuit will have their normal values complemented. Based on the failure information of test vectors, we use the method in Section IV and V to distinguish between the suspected scan cells.

For each circuit in Table 3, , we report the fraction of times at least one component of the injected multiple chain faults appears in the top-N (N=1,2,5) suspect list (@2 means that at least one chain fault is reported in top-2 suspect list). The entries in the table represent the accuracy for 100 trials. In the experiment, the parameter *margin* in the method of Section IV.A is 0.5. From the result, we can see that the proposed method can obtain near 100% accuracy with dozens of logic defects, when there is one chain fault. Even for two or three chain faults, the accuracy is very high with dozens of logic defects. The time of the flush test in our method is about six times as that for linear scan chains. The average CPU runtime (9 cases for each circuit) for the five circuits are 1.9, 3.4, 10.5, 11.7 and 7.2 seconds in one trial, respectively. Since the time is mainly consumed by simulation process, highly parallel solutions are possible when used in the industry for large circuits. To the best of our knowledge, this is the first scan-chain diagnosis method that can tolerate so many system logic defects without a significant diminishing of its performance.

VII. Conclusions

In this paper, we proposed a combined hardware-software based approach to scan-chain diagnosis, in the presence of system logic defects. First, we used flush patterns to narrow down the suspected scan cells to a small number. Next, we improved the resolution for timing faults to one with the help of another flush test. Further improvements in the resolution of stuck-at faults were made. The proposed approach was extended to multiple faults in a scan chain validated for its accuracy by experiments on benchmark circuits.

References

[1] Y. Huang, R. Guo, W.-T. Cheng, and J. C.-M. Li,"Survey of Scan Chain Diagnosis," IEEE Design & Test of Computers, vol. 25(3), 2008, pp. 240-248.

[2] Y. Huang, W. Hsu, Y.-S. Chen, W.-T. Cheng, R. Guo, A. Mann, "Diagnose Compound Scan Chain and System Logic Defects," Proc. of. Int'l Test Conf., 2007, pp. 1-10.

[3] F. Wang, Y. Hu, Y. Huang, H. Li, X. li, J. Ye, "Deterministic Diagnostic Pattern Generation (DDPG) for Compound Defects", Proc. of. Int'l Test Conf., 2008, paper 14.1.

[4] R. Guo and S. Venkataranman, "A Technique for Fault Diagnosis of Defects in Scan Chains," Proc. Int'l Test Conf., 2001, pp. 268-277.

[5] Y. Huang et al., "Statistical Diagnosis for Intermittent Scan Chain Hold-Time Fault," Proc. Int'l Test Conf., 2003, pp. 319-328.

[6] C-W Tzeng, J-S Yang, and S-Y Huang, "A Versatile Paradigm for Scan Chain Diagnosis of Complex Faults Using Signal Processing Techniques,", ACM Trans. Design Automation of Electronic Systems, 13(1), January 2008, pp. 9:1-9:27.

[7] Z. Chen, S Seth, D. Xiang, and B. B. Bhattacharya, "PVT: Unified Reduction of Test Power, Volume, and Test Time using Double-Tree Scan Architecture," Journal of Low Power Electronics (JOLPE), 6(3), October 2010, pp. 457-468.

[8] P. Song et al., "A Novel Scan Chain Diagnostics Technique Based on Light Emission from Leakage Current," Proc. Int'l Test Conf. (ITC 04), IEEE CS Press, 2004, pp. 140-147.

[9] F. Stellari et al., "Broken Scan Chain Diagnostics Based on Time-Integrated and Time-Dependent Emission Measurements," Proc. 30th Int'l Symp. Testing and Failure Analysis (ISTFA 04), 2004, pp. 52-57.

[10] S. Narayanan and A. Das, "An Efficient Scheme to Diagnose Scan Chains," Proc. Int'l Test Conf. (ITC 97), IEEE CS Press, 1997, pp. 704-713.

[11] F. Motika, P.J. Nigh, and P.T. Tran, Diagnostic Method for Structural Scan Chain Designs, US patent 6961886, Patent and Trademark Office, 2005.

[12] Y.-L. Kao, W.-S. Chuang, J. C.-M. Li, "Jump Simulation: A Technique for Fast and Precise Scan Chain Fault Diagnosis," Proc. Int'l Test Conf. (ITC), 2006, pp. 1-9.

[13] R. Guo, Y. Huang, and W.-T. Cheng, "A Complete Test Set to Diagnose Scan Chain Failures," Proc. Int'l Test Conf. (ITC 07), IEEE Press, 2007.

[14] A. Crouch, "Debugging and Diagnosing Scan Chains," EDFAS, Vol. 7, No. 1, 2005, pp. 16-24.

[15] J.C.-M. Li, "Diagnosis of Multiple Hold-Time and Setup-Time Faults in Scan Chains," IEEE Trans. Computers, vol. 54, no. 11, Nov. 2005, pp. 1467-1472.

[16] J.-S. Yang and S.-Y. Huang, "Quick Scan Chain Diagnosis Using Signal Profiling", Proc. of Int'l Conf. On Computer Design, Oct., 2005, pp. 157-160.

[17] N. Schemm, S. Balkir, and S. Seth, "Hardware implementation of the double-tree scan architecture," in Proc. Int. Symposium on Circuits and Systems, 2010.

[18] B. B. Bhattacharya, S. C. Seth, and Z. Sheng, "Double-tree scan: a novel low-power scan-path architecture," in Proceedings Int'l Test Conf., 2003, pp. 470-479.