Built-in-Self-Test of FPGAs with Provable Diagnosabilities and High Diagnostic Coverage with Application to On-Line Testing

Shantanu Dutt, *Member, IEEE,* Vinay Verma and Vishal Suthar

***Abstract*— We present novel and efficient methods for built-in-self-test (BIST) of FPGAs for detection and diagnosis of permanent faults in current as well as emerging technologies that are expected to have high fault densities. Our basic BIST methods can be used in both on-line as well as off-line testing scenarios, though we focus on the former in this paper. We present 1- and 2-diagnosable BISTer designs that make up a ROving TEster (ROTE). Due to their provable diagnosabilities, these BISTers can avoid time-intensive adaptive diagnosis without significantly compromising *diagnostic coverage*—the percentage of faults correctly diagnosed. We also develop functional testing methods that test programmable logic blocks (PLBs) in only two circuit functions that will be mapped to them as the ROTE moves across a functioning FPGA. We extend our basic BISTer designs to those with test-pattern generators (TPGs) using multiple PLBs to more efficiently test the complex PLBs of current commercial FPGAs, and prove the diagnosabilities of these designs as well. Simulation results show that our 1-diagnosable functional-test based BISTer with a 3-PLB TPG has very high diagnostic coverages—for example, for a random fault distribution, our non-adaptive diagnosis methods provide diagnostic coverages of 96% and 88% at fault densities of 10% and 25%, respectively, while the previous best non-adaptive diagnosis method of the STAR-3×2 BISTer has diagnostic coverages of about 75% and 55% at these fault densities.**

***Index Terms*— built-in self-test (BIST), k-diagnosability, diagnostic coverage, FPGAs, functional testing, on-line testing, roving tester.**

I. Introduction

An FPGA consists of an array of programmable logic blocks (PLBs) interconnected by a programmable routing network and programmable I/O PLBs. Current technology trends for FPGA devices are in the very deep-submicron (VDSM) regime with recent chips using 90 nanometers and seven metal layers. Unfortunately, this trend has resulted in decrease of fabrication yield, and can potentially lead to decreased reliability of operation. The larger die sizes also mean that there is more likelihood of failure of some component. Thus testing and fault tolerance techniques for FPGAs are important to increase device fabrication yield, and the reliability of FPGA operation in platforms ranging from mission/life-critical systems to commercial products. For current very deep submicron CMOS technology, and to a greater extent for emerging molecular nano-technology and nanoscale CMOS technology, it is very likely that transient faults will disturb the system operation and also permanent faults will develop during the FPGA's operational lifetime. The rate of occurrence of permanent faults can be quite high in emerging technologies, and hence there is a need for periodic testing of such FPGAs.

The BIST methods presented in this paper are mainly targeted at detection and diagnosis of *permanent faults* that model fabrication defects as well as other physical defects arising during the lifetime of the FPGA chip. We will use the term *fault density* to mean the percentage of PLBs that have a permanent fault. Finally, we define *diagnostic coverage* of a testing technique as the percentage of faults it correctly diagnoses, and its *fault latency* as the average time it takes from the start of the testing phase for the FPGA to diagnose a fault.

Off-line testing methods for FPGAs are reasonably mature and well developed [2], [10], [11]. Off-line testing is acceptable in application environments where there are little real-time constraints on the application circuit mapped to the FPGA, since such testing requires the circuit to stop functioning while it is tested. However, in systems with such constraints like those in space, avionics and many commercial products, it is desirable and sometimes necessary to perform testing *on-line*, i.e., testing with the application circuit mapped to and executing on the FPGA and with minimal disruption to its functioning. As mentioned earlier, the rate of occurrence of permanent faults can be quite high in emerging technologies. Hence there is a need for on-line testing to frequently monitor and check such FPGAs; this would be especially crucial in remote, mission-critical and other critical applications. The periodicity of on-line testing can be adjusted to the susceptibility (fault probability) of the FPGA technology. While the enveloping techniques we present here are for on-line testing, the basic BIST methods we develop can be readily used for off-line testing as well; in fact, the higher diagnosability of our BISTers should also improve diagnostic coverage in the off-line testing scenario compared to previous work. In the off-line testing mode, instead of having a roving tester (see Sec. II), multiple copies of our BISTers would be configured into the entire FPGA to perform testing in parallel, thereby

S. Dutt is with the University of Illinois at Chicago (UIC), and V. Verma and S. Suthar are with Xilinx Inc. S. Suthar was at UIC when this paper was submitted.

This work was funded by a grant from Xilinx Corp., Darpa Contract # F33615-98-C-1318, and National Science Foundation under Grant No. 0204097.

yielding smaller test times[1].

We differentiate between *exhaustive* and *functional* testing of PLBs. In the former, a PLB is tested in all its modes of operations, i.e., for all combination of values in its lookup tables (LUTs), all possible settings of its flip-flops (FFs), etc. This will be required when it is not known which circuit(s) will be mapped to the FPGA, as, e.g., during factory-testing of FPGAs that could be sold to any user. In functional testing, which can only be done when it is precisely known which (small) set of circuits will be mapped to the FPGA, a PLB is tested with it configured in only those functional modes in which it will be used at different times across these circuits. For simplicity of exposition, we assume though that for functional testing only one circuit will be mapped to the FPGA. Functional testing can be done in the field when the user-circuit is generally known. It can also be used for factory testing when it is known which application will be mapped to the FPGA as, e.g., for FPGAs to be used in printers from a certain vendor. Functional testing at the factory allows FPGAs with faults that do not affect the correct execution of the application circuit to be mapped to it to be correctly diagnosed as 'OK' thus increasing chip yield and reducing costs. In this paper, we present on-line methods for both exhaustive as well as functional testing of FPGAs.

Before proceeding further, we give a useful definition. A testing technique is said to be *k-diagnosable* if in the presence of any $m \leq k$ faulty components it can correctly identify all m faulty components among the $n \geq k$ components that it tests. Such a testing technique is equivalently said to have a *diagnosability* of k.

There have only been a few methods proposed for on-line FPGA testing, including [1][2], [3], [4], [16], [18] In [16], Shnidman et al. proposed a fault-scanning technique for testing a portion of an FPGA; for the technique to work, a bus-based non-segmented interconnect architecture is assumed that is not available in mainstream commercial FPGAs. The method of [4] is among the first that uses a roving tester called STAR to test a portion of the chip exhaustively (irrespective of the mapped circuit) while the rest of the chip executes the application circuit. Their tester, however, has no provable diagnosability and in fact as we show later in Sec. III, the basic built-in self tester (BISTer) in [4] is *0-diagnosable*, i.e., it cannot identify the single faulty PLB. Furthermore, since it uses an adaptive diagnosis method to get around the 0-diagnosability problem, and also performs exhaustive testing of PLBs (even when the circuit is known), it has high test times and thus high fault latencies. In [1], [3], a 3×2 BISTer was proposed in combination with the STAR tester of [4] that provides 1-diagnosability among 6 PLBs (i.e., if there is at most one fault among 6 PLBs, the faulty PLB will be correctly

[1]To improve diagnosability, two such FPGA-wide multiple-BISTer configurations can be used, with the second one's configuration shifted two columns to the right of the first one (with possible wrap-around of the rightmost BISTers if they are more than 2-column wide).

[2]After the submission of the first draft of this paper, and during its review cycle, [1] was published, and this was pointed out to us by one of the reviewers. Paper [1] is an extension of [3] with the main addition being the Multicello fault-diagnosis algorithm for an extended version of their BISTer with a 4×2 tile that are discussed in Sec. III-C.

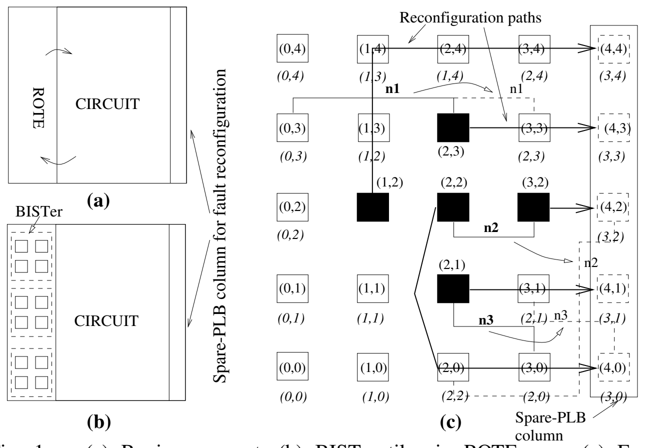


Fig. 1. (a) Roving concept. (b) BISTer tiles in ROTE area. (c) Four reconfiguration paths (shown by dark arrows) for four faulty PLBs and net re-routings (shown by dashed lines) of nets connected to PLBs on these paths. The new PLB labels in the reconfigured FPGA are shown in italics below each PLB (previous labels are in roman); PLB with new/italics label (i,j) functionally replaces the previous/roman (i,j)-labeled PLB.

identified); it also diagnoses some but not all patterns of two PLB faults. This BISTer, which we call STAR-3×2 BISTer, will be discussed further in Sec. III. BISTers for interconnects are presented in [18], [19].

We present here a roving tester (ROTE) for on-line testing of FPGAs, which tests parts of the circuit in a piecemeal manner by duplication and comparison using BISTers with *provable diagnosabilities*, thus avoiding expensive adaptive diagnosis. The main contribution of this work, which is a significant extension of [20], are novel 1- and 2-diagnosable BISTer designs, functional testing methods coupled with the 1-diagnosable BISTer, extensions of these designs for application to many current FPGAs, and diagnosability results for all these designs. Our testing methods are more accurate (higher diagnosability and higher diagnostic coverage[3]) and faster than those developed in previous work. In this paper we address PLB faults; interconnect fault test-and-diagnosis has been addressed in [19].

The rest of the paper is organized as follows. Section II gives a bird's-eye view of the on-line testing and fault tolerance environment that we propose; only on-line testing is discussed in the rest of the paper. In Sec. III we discuss previous BISTer designs for on-line testing—we establish that a previous BISTer design [4] is 0-diagnosable, and also present two 1-diagnosable BISTers [1], [3], one with a 3×2 tile and another with a 4×2 tile. Section IV presents two new BISTer architectures with provable diagnosabilities of one and two, while Sec. V develops a technique for faster testing and diagnosis that tests PLBs in only two functional modes that they can be used in, in an operational FPGA with the roving tester. In Sec. VI, we extend out conceptual BISTer designs to perform testing in current FPGAs that have PLBs with large functionalities in terms of the number of inputs. Section VII presents our simulation results, and we conclude in Sec. VIII.

[3]Note that *diagnosability* is a measure of the diagnosis capability of a BISTer within a single BIST area, while *diagnostic coverage* is an average-case measure of diagnosis capability of a testing technique across a system (FPGA chip in our case) wherein it employs multiple BISTers to cover the entire system. The two measures are clearly correlated.

II. The Big Picture – Roving Tester and Fault Reconfiguration

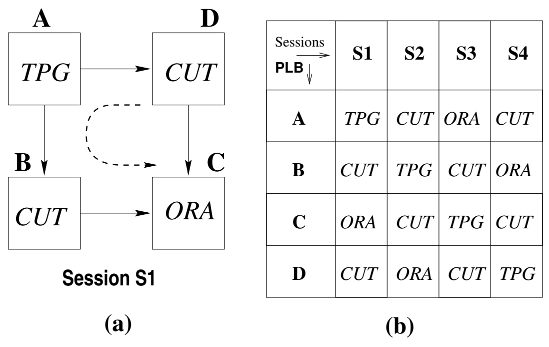
We present here the overview of the process in which the FPGA is repeatedly tested in the field in an on-line manner for faults by a roving tester, and on whose detection the application circuit mapped to the FPGA is dynamically reconfigured; if at any point in this process a fault is non-reconfigurable, then the system reports an "irrecoverable failure". This scenario mainly applies to on-line field testing. An earlier work [4] presented the concept of a roving tester; our roving tester, however, has a different structure and roving mechanism. Note that the main contribution of our paper is not the roving mechanism or in general the roving tester, but the BISTers that comprise the roving tester.

We consider SRAM-based FPGAs that support partial and run-time reconfiguration such as Xilinx's Virtex series. In an $n \times n$ FPGA, two columns of the FPGA are left spare for the ROTE and, say, one spare column is allocated to the right of the FPGA for fault reconfiguration. The system function is implemented in the remaining $n \times (n-3)$ subarray; see Fig 1(a). The leftmost spare columns are occupied by a roving tester (ROTE) that roves across the FPGA and performs test and diagnosis. The ROTE area comprises of multiple BISTers that simultaneously test different sub-areas of the ROTE; see Fig 1(b). An external reconfiguration and test controller controls the BISTer operation (configures the BISTers in the ROTE area, starts their operations, scans out their detailed syndromes, and performs fault diagnosis as explained in this paper), the movement of the ROTE across the FPGA and fault reconfiguration. The new functions of the PLBs and the new track positions of the nets for moving the ROTE to its next position one column to the right are computed concurrently by the controller while the ROTE performs testing in its current position.

Fault detection in a BISTer consists of comparing the output response of a PLB to another identically configured PLB; a mismatch in the output response indicates faulty PLBs in the BISTer tile. *Diagnosis* includes drawing inferences from the output vectors and locating the exact faulty PLB(s). Once the testing in the ROTE area is over, the circuit functioning is stopped momentarily and the new bit-streams are downloaded to the FPGA by the controller to move the ROTE to the right by one column. In the presence of faults, the ROTE will move in a warped manner so that, say, each BISTer-1 tile (described in Sec. IV-A) in it occupies a 2×2 array of PLBs whose latest diagnosis status is "fault-free". This is shown in Fig. 9 (this figure also illustrates Lemma 1 in Sec. V).

When fault(s) are detected and diagnosed, then using techniques proposed in [7], [14], each faulty PLB is reconfigured directly or indirectly using a unique spare PLB. Briefly, the steps are:

1. Compute *reconfiguration paths* from each identified faulty PLB to a spare PLB using fast network flow algorithms presented in [14]. PLB v following PLB u in a reconfiguration path means that v will be configured with u's functionality for each such (u, v) pair to achieve reconfiguration; see Fig. 1(c).
2. Perform incremental re-routing (e.g., [6], [7]) to extend/re-route each interconnect going originally to u to now connect to v for each adjacent (u, v) pair in a reconfiguration path. Figure 1(c) shows three nets n_1, n_2, n_3 of the original configuration and their re-routings required (shown by dashed lines) for reconfiguration.



Sessions → / PLB ↓	S1	S2	S3	S4
A	*TPG*	*CUT*	*ORA*	*CUT*
B	*CUT*	*TPG*	*CUT*	*ORA*
C	*ORA*	*CUT*	*TPG*	*CUT*
D	*CUT*	*ORA*	*CUT*	*TPG*

Fig. 2. (a) BISTer-0 architecture of [4]. (b) Cycling of PLBs in BISTer-0 tile.

III. Previous BISTer Designs

In this section, we first present the BISTer design of [4] denoted here by BISTer-0, and prove that it is 0-diagnosable. While BISTer-0 has been superseded by the 1-diagnosable 3×2 BISTer of [1], [3] by the same primary designers of BISTer-0, the latter provides a starting point for us for developing our own 1-diagnosable BISTer that has better diagnostic coverage than the 3×2 BISTer of [1], [3] as we show in Sec. VII. We then also present the design of the 3×2 BISTer of [1], [3] (henceforth called the STAR-3×2 BISTer), and its (4×2)-tile version along with its diagnosis algorithm Multicello.

A. A 0-diagnosable BISTer

The presentation of BISTer-0 sets the stage for developing new BISTer designs that have provable diagnosability in the next section. Throughout the paper we assume that interconnects and wires are fault free; testing and diagnosis of faulty interconnects has been addressed in [18], [19].

BISTer-0 shown in Fig. 2 comprises of one test pattern generator (TPG), one output response analyzer (ORA) and two PLB cells under test (CUTs) that are exhaustively tested. The TPG applies test patterns to two identically configured CUTs whose outputs are compared by the ORA. The ORA latches and reports mismatches as test failures. The testing of the two CUTs by the TPG and ORA in one BISTer configuration is called a *session*. Bister-0 has four sessions and successive sessions are obtained by one-PLB rotations of the BISTer functions (two CUTs, TPG, ORA), shown by a dotted arc in Fig. 2a. Figure 2a also shows the configuration for session S_1, and Fig. 2b shows all the sessions of BISTer-0. In session S_2, PLB A becomes a CUT, PLB B becomes a TPG, PLB C becomes a CUT and PLB D becomes an ORA. When the BISTer completes a cycle of four sessions, each PLB has been configured twice as a CUT. BISTer-0 can detect multiple faults with high probability [4] but, as we show later, it cannot diagnose, i.e., locate, any of them—it is 0-diagnosable. Thus adaptive diagnosis schemes, which are generally time consuming, are used in [4] even for a single PLB fault.

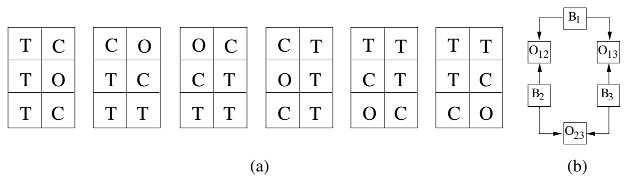


Fig. 3. (a) Six sessions of STAR-3 × 2 BISTer [1], [3] (T ≡ TPG, C ≡ CUT, O ≡ ORA). (b) One of its two "combined test sessions" represented by an ORA test graph.

We define the *detailed syndrome* for a session as the *0/1 bit pattern* observed at the ORA output over all test vectors of the TPG; a *0* indicates a match and a *1* indicates a mismatch. S_i represents the *i'th* session as well as the detailed syndrome for S_i*'th* session, the use of which will be clear from the context.

The *gross syndrome* of a session is the overall *pass/fail* observation over all modes of tested operations for that session. In other words, the gross syndrome of a session is a "X" (*fail*) if the ORA output is "1" for any input test vector, and is a "$\surd$" (*pass*), otherwise.

Theorem 1: BISTer-0 is zero-diagnosable.

Proof: There are four sessions for BISTer-0; see Fig. 2b. In BISTer-0 the same pair of PLBs are configured as CUTs in two different sessions. When either PLB fails, the gross and detailed syndrome will be identical in both sessions in which they are configured as CUTs thus making it impossible to locate the fault in either of them. For example, if PLB A fails as a CUT only (i.e., its fault is detected when it is tested in all its modes when it is a CUT, but this fault is not exercised when it is configured as a TPG or an ORA), then the gross syndrome of sessions S_2 and S_4 will be *fail*, while S_1 and S_3 will be a *pass*. The same syndrome will be obtained when C is faulty as a CUT only; see Fig. 2b. Also A and C failing in the same mode that is not exercised by the TPG or ORA will also produce the same detailed syndromes for all sessions. Thus in such cases, we cannot determine if A or C is faulty. Similarly, we cannot distinguish between faulty PLBs B and D. Hence BISTer-0 is zero-diagnosable. $\diamond$

B. The 1-diagnosable STAR-3×2 *BISTer*

We now briefly present the 1-diagnosable STAR-3 × 2 BISTer of [1], [3] to which we directly compare in Sec. VII a version of our 1-diagnosable BISTer with a 3-PLB-TPG, BISTer-$1^{2\times3}$ (see Sec. VI-B) w.r.t. diagnostic coverage and fault latency metrics.

As shown in Fig. 3, the STAR-3 × 2 BISTer has a 3 × 2 arrangement and has a 3-PLB TPG in order to simultaneously test all lookup tables (LUTs) in a PLB with input-output pin ratio of 3:1 (in Sec. VI we further discuss this and alternative strategies of tackling input-output pin ratios of greater than 1). The STAR-3×2 BISTer has six sessions, and each session is obtained by a rotation of BISTer functions of the previous session. The goal of the six sessions is to test each PLB twice and compare it to a different PLB each time it is tested. The rotating strategy leads to this BISTer being 1-diagnosable (out of 6 PLBs). The diagnosis technique (derived from Theorem 1 of [3]) here is that when the gross syndrome consists of exactly one pair of ORAs with a common CUT reporting faulty syndromes (across all six sessions), and in which the common CUT when an ORA may or may not report a faulty syndrome, then the common CUT is uniquely diagnosed as being faulty (assuming there are no other faulty PLBs). E.g., if the ORAs in the first and third sessions of Fig. 3(a) report failures, and among the other ORAs in the remaining four sessions, at most the ORA in the second session reports a failure, then the top CUT in the first session is diagnosed as faulty. This is the non-adaptive diagnosis technique of [3] that we use in our simulation of the STAR-3×2 BISTer. As claimed in [1], the STAR-3 × 2 BISTer can also detect (though not diagnose) 2-PLB fault patterns under simplifying assumptions of a TPG with a faulty PLB not skipping test vectors that detect fault(s) in a faulty CUT, or the two faulty PLBs not having the same responses to all test vectors (common-mode failures)[4].

The more involved diagnosis scheme Multicello presented in [1] corresponds to their 4×2 BISTer, and is extended from its off-line version [2]. We discuss both, their 4 × 2 BISTer and Multicello, below.

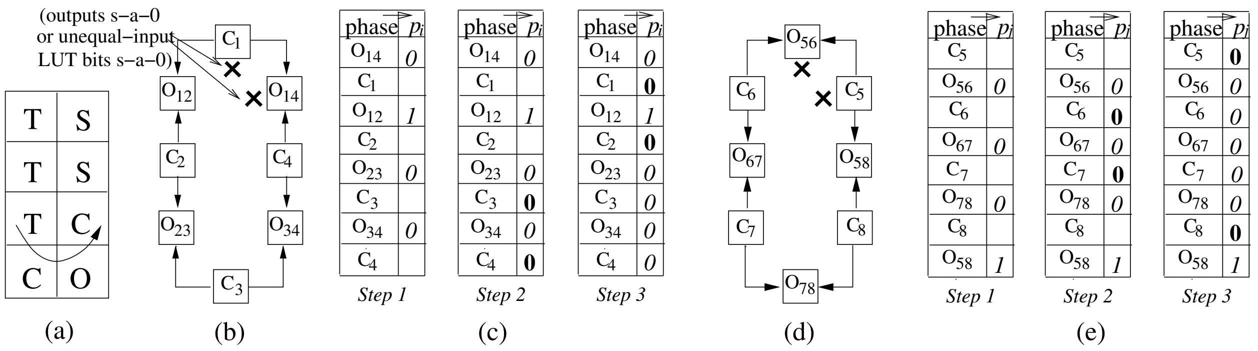
C. The 1-diagnosable STAR-4×2 *BISTer*

Figure 4(a) shows the STAR-4 × 2 BISTer, which has a 4 × 2 tile and is similar in structure to the STAR-3 × 2 BISTer except for two consecutive "spare" PLBs (labeled 'S') embedded in the tile of the former. Since there are 8 PLBs in this BISTer, the functional rotations, including that of the "spare functionalities", yield 8 sessions. The 8 sessions can be partitioned into two "combined test sessions", each of which consists of four non-consecutive sessions. In each combined test session, disjoint sets of PLBs are CUTs, and in each, the comparison results of each CUT are reported by two different ORAs. The *ORA test graphs* for the two combined test sessions representing the aforementioned relationship between CUTs and ORAs are shown in Figs. 4(b) and (d); O_{ij} in an ORA test graph is used to denote the ORA that reports the comparison results of CUTs C_i and C_j.

The fault diagnosis scheme Multicello [1] for the 4 × 2 BISTer attempts to diagnose multiple faults as follows (refer to the diagnosis table of Fig. 4(c) for examples of the given steps):

1. Record ORA results (0 for a pass gross syndrome, 1 for a fail) and initialize the failure state of every CUT in each phase as unknown (empty)—a *phase* is a specific functional configuration of the CUT PLBs.
2. In each column, for every two consecutive ORAs with a 0 mark, enter a 0 for the CUT between them.
3. In each column, for every two adjacent 0 marks followed by an empty cell, enter a 0 in the empty cell.
4. In each column, for every adjacent 0 and 1 marks followed by an empty cell, enter a 1 in the empty cell.
5. Consistency checks: If there is an ORA reporting a failure in phase p (marked with a 1), while neither of the two CUTs

[4]In Theorem 4 we also make some simplifying assumptions for proving the 2-diagnosability of BISTer-2 (Sec. IV-B), which is a much higher capability than 2-detectability. However, our assumptions allow for the faulty PLBs to malfunction when they are TPGs (e.g., skip fault-detecting test vectors for the other faulty PLB when it is a CUT) or ORAs, and does not have the requirement of no common-mode failures in the two faulty PLBs. Specifically, our assumptions are that both faulty PLBs either malfunction as TPGs and ORAs or correctly perform as TPGs and ORAs.



phase $\xrightarrow{}$	p_i (Step 1)	p_i (Step 2)	p_i (Step 3)
O_{14}	*0*	*0*	*0*
C_1			**0**
O_{12}	*1*	*1*	*1*
C_2			**0**
O_{23}	*0*	*0*	*0*
C_3		**0**	*0*
O_{34}	*0*	*0*	*0*
C_4		**0**	*0*

phase $\xrightarrow{}$	p_j (Step 1)	p_j (Step 2)	p_j (Step 3)
C_5			**0**
O_{56}	*0*	*0*	*0*
C_6		**0**	*0*
O_{67}	*0*	*0*	*0*
C_7		**0**	*0*
O_{78}	*0*	*0*	*0*
C_8			**0**
O_{58}	*1*	*1*	*1*

Fig. 4. (a) One session of STAR-4 × 2 BISTer [1], [3] (T ≡ TPG, C ≡ CUT, O ≡ ORA, S ≡ Spare). (b) Its first combined test sessions represented by its ORA test graph; the two faulty PLBs are indicated by X's. (c) Diagnosis using the non-adaptive phase (Steps 1-4) of Multicello for the first combined test session for the two faults shown—Step 4 is not needed as all locations in the table are filled in the first three steps, and no CUTs are diagnosed as faulty. (d) The ORA test graph for the second combined test session. (e) Its diagnosis for the two faults using non-adaptive Multicello—again, Step 4 is not needed, and no faults are diagnosed.

observed by the ORA fails in that phase (both are marked with a 0), then there is a potential inconsistency. If the failing ORA is reported as faulty in the other combined test session where it is a CUT, then go to Step 6. Otherwise, divide the suspect PLBs into subsets, retest and reapply the procedure to each subset. If no further division is possible, then report inconsistency and exit.

6. If every PLB has been identified as fault-free or faulty, the group of faulty PLBs has been uniquely diagnosed. Otherwise, divide the suspect PLBs into subsets, retest and reapply the procedure to each subset.

For more than a single faulty PLB, the two assumptions made by Multicello and the theoretical diagnosis results it relies on are [1], [3]:
(1) Assumption A1: A TPG with faulty PLBs does not skip the patterns that detect faults in a CUT.
(2) Assumption A2: No more than two faulty CUTs have identical responses in the same failing phase.
However, the theoretical diagnosis results (Lemmas 2-8 and Theorem 2 of [1]) for various gross syndrome patterns used by Multicello do not take into consideration the possibility of ORA faults. These results thus do not hold when specific ORA faults occur (within the constraints of the above two assumptions). Multicello accounts for situations like this (called "inconsistent syndromes"—those that do not conform to the syndrome patterns specified in their theoretical results) by using adaptive diagnosis; see Steps 5 and 6 of Multicello. Thus, while the Multicello procedure can diagnose some patterns of multiple faults correctly in one pass, there is no theoretical guarantee that it can diagnose all such patterns (including all 2-fault patterns) without resorting to adaptive diagnosis. Another issue in Multicello is that the definition of which PLBs can be considered as "suspect" (see Steps 5 and 6 of Multicello) in different fault syndrome classes has not been specified except for one case[5]. Below, we provide two examples of 2-fault patterns that cannot be diagnosed by Multicello without using adaptive diagnosis; one of these fault patterns is, in fact, mis-diagnosed by the non-adaptive phase (Steps 1-4).

In the first combined test session shown in Fig. 4(b), consider the two consecutive faulty PLBs C_1 (which becomes the ORA O_{56} in combined test session 2 of Fig. 4(d)) and $O_{1,4}$ (which becomes the CUT labeled C_5 in combined test session 2), each having either of the following faults: (i) either their output(s) are stuck-at-0, or (ii) their LUT bits corresponding to unequal inputs (from the two CUTs they would compare as ORAs) are stuck-at-0. Either of these internal faults causes these PLBs when configured as ORAs to always report a pass (0) output, even with mismatched inputs. Assume further that these PLBs, while malfunctioning in the same way as ORAs, have different internal faults (e.g., one has fault (i) and the other (ii)), so that as CUTs they do not have identical outputs in all phases of testing. Figures 4(c) and (e) shows the results of the application of the first three steps of Multicello (Step 4 is not needed as all positions get filled before it is reached), for relevant phases p_i and p_j (in which fault syndromes are reported by the ORAs). As can be seen, no CUT is diagnosed as faulty. However, the inconsistent syndrome check of Step 5 is triggered, and each PLB in the suspect set needs to be configured in different BISTers to diagnose them further in an adaptive phase. Since the suspect set for this fault syndrome class is not defined, we can only surmise that at the very least, the suspect set includes the PLBs that are checked by the ORAs that report fail syndromes, as well as those ORAs (since it is possible that the ORAs are reporting fail syndromes because of internal faults, e.g., their outputs are stuck-at-1). This gives us 4 PLBs as suspect, and diagnosing them further with 4 different BISTers, whose other PLBs are not guaranteed to be fault-free, could be involved.

Figure 5 shows another example with PLBs corresponding to ORAs O_{12} and O_{14} faulty with identical responses of an output of 1 when they are configured as ORAs, even when they receive matching inputs from the two CUTs they compare. Assume also that their faults are such that they produce identical responses as CUTs in any phase (e.g., both their outputs are stuck-at-1 or LUT bits corresponding to identical/matching inputs [which, when configured as ORAs, they would receive from fault-free CUTs] are stuck-at-1).

[5]The only exception is Lemma 7 of [1] which states "(For the 4 × 2 tile) If only two ORAs without a common CUT fail phase p, then at least one pair of CUTs between the two ORAs are faulty and have identical response in phase p." This lemma implicitly identifies the set of suspect PLBs (though that term is not used there) as the set of 4 PLBs checked by the two ORAs in question. Other than this case, no other suspect PLB sets are identified for any other fault syndrome patterns.

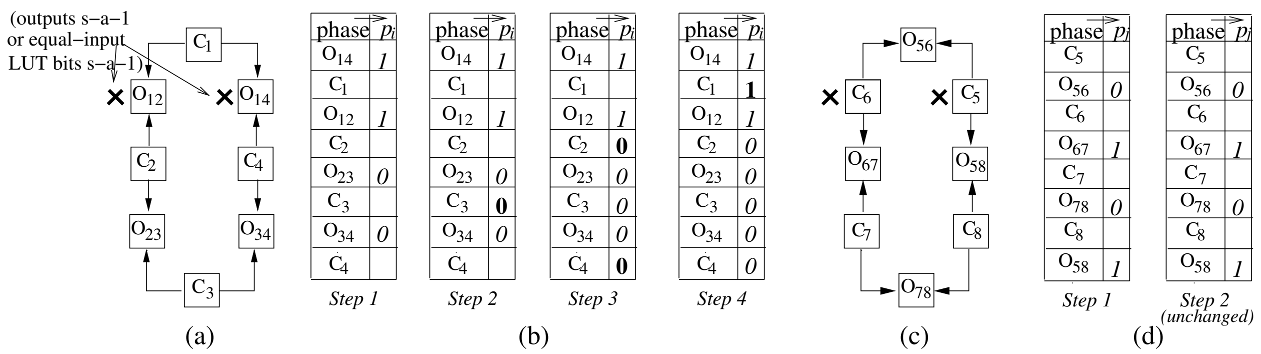


Fig. 5. (a) The ORA test graph for the first combined test session, with two faulty PLBs indicated by X's. (c) Its diagnosis for the two faults using the non-adaptive phase (Steps 1-4) of Multicello for the first combined test session—C_1 is mis-diagnosed as faulty. (d) The ORA test graph for the second combined test session. (e) Its diagnosis for the two faults using non-adaptive Multicello—no CUT locations can be filled and thus no fault diagnosis is possible.

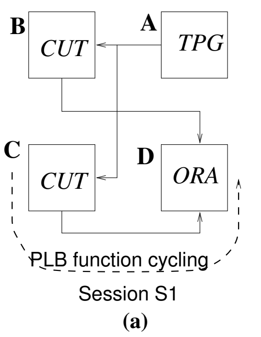
Figure 5(b) shows the application of Steps 1-4 of Multicello for the combined test session represented in Fig. 5(a). This results in a mis-diagnosis of C_1 as faulty. For the second combined test session represented in part (c), Multicello cannot proceed beyond Step 2 and no diagnosis is performed. Again, the inconsistent syndrome check of Step 5 is triggered. In this case, at least 6 PLBs are suspect, and if PLB C_1 diagnosed as faulty in the first combined test session is also included, as it should be, 7 PLBs are in the suspect set, requiring 7 different BISTers to diagnose them.

The above two examples show that Multicello is a 1-diagnosable technique, and for multiple faults, involved adaptive diagnosis may be required in some cases. As explained above, the adaptive diagnosis phase of Multicello is used when it encounters inconsistent syndromes than can occur when there are more than one fault in the BIST area. This also seems to be the case for the adaptive diagnosis phase for STAR-3$\times$2 BISTer given in [3]. This phase uses a divide-and-conquer strategy in which the "suspected" PLBs in the original BISTer are re-configured into different BISTers for further test and diagnosis, and can require several reconfigurations of the FPGA tester area. Moreover, there are no results on the convergence of this adaptive phase.

As established in Sec. VI-B, our 3-PLB-TPG BISTer, BISTer-$1^{2\times3}$, while also being 1-diagnosable out of 6 PLBs (like STAR-3 $\times$ 2 BISTer), can provably diagnose other fault patterns using only non-adaptive diagnosis. Furthermore, as we empirically show in Sec. VII, BISTer-$1^{2\times3}$ has higher diagnostic coverage than the STAR-3 $\times$ 2 BISTer using the provable non-adaptive 1-diagnosable technique of [1], [3].

IV. New BISTer Architectures

We now present our new BISTer designs that have non-zero diagnosis capabilities—BISTer-1 which is 1-diagnosable and BISTer-2 which is 2-diagnosable with very high probability. Note that non-zero diagnosability of the BISTer reduces fault latency since time-consuming adaptive diagnosis procedures such as those used in [4] will not be needed as long as the number of faults is within the diagnosability of each BISTer tile; these numbers are 1 fault out of 4 PLBs for BISTer-1 or a 25% fault density, and 2 faults out of 6 PLBs in BISTer-2 or a 33% fault density. In this section, we present the



Sessions → / PLB ↓	S1	S2	S3	S4
A	*TPG*	*ORA*	*CUT*	*CUT*
B	*CUT*	*TPG*	*ORA*	*CUT*
C	*CUT*	*CUT*	*TPG*	*ORA*
D	*ORA*	*CUT*	*CUT*	*TPG*

(b)

Fig. 6. (a) Our BISTer-1 architecture. (b) The 4 sessions of BISTer-1.

BISTer designs for exhaustive PLB testing. In the next section we present the basic BISTer-1 design with a modified test-and-diagnosis envelope for functional PLB testing; a similar approach can be used for BISTer-2. Finally, we assume here without loss of generality that a TPG can be configured in one PLB. In Sec. VI we provide detailed discussions on how a one-PLB TPG can be used to test other PLBs when the PLB input/output pin ratio is greater than one, and alternatively, how to extend BISTer-1 to a BISTer with a 3-PLB TPG that can also diagnose one fault besides other fault patterns.

A. A New 1-diagnosable BISTer

Figure 6 shows a modified BISTer architecture BISTer-1. Like BISTer-0, it consists of one TPG, two CUTs and an ORA, however, unlike in BISTer-0 where two diagonally opposite PLBs are configured as CUTs, in BISTer-1 two adjacent PLBs are CUTs. This results in a PLB being a CUT in two consecutive sessions and each pair of PLBs beings CUTs in exactly one session. In contrast, in BISTer-0 there are two pair of PLBs that are each CUTs in two different sessions. This difference is key to providing 1-diagnosability in BISTer1.

As shown in Fig. 6, in session S_1, PLB A is configured as a TPG, *B* as a CUT, *C* as a CUT and *D* as an ORA. Again, successive sessions are obtained by one-PLB rotations of the BISTer functions.

In Sec. VI-B, we extend BISTer-1 to a 3-PLB-TPG BISTer, BISTer-$1^{2\times3}$, for simultaneously testing all lookup tables (LUTs) of large PLBs with input/output pin ratios of up to 3:1. There we also establish the diagnosabilities of BISTer-$1^{2\times3}$. Note that BISTer-1 can test the LUTs of the latter type of PLBs sequentially with its 1-PLB TPG assuming the number

TABLE I

GROSS SYNDROMES FOR BISTER-1 AND THEIR DIAGNOSIS UNDER ASSUMPTION OF AT MOST ONE FAULTY PLB.

Ses. → S.No ↓	S_1	S_2	S_3	S_4	Inference
1	√	√	√	√	No faulty PLB
2	X	√	√	√	Fault not in PLB
3	√	X	√	√	Fault not in PLB
4	√	√	X	√	Fault not in PLB
5	√	√	√	X	Fault not in PLB
6	X	X	√	√	Faulty C *(CUT)*
7	√	X	X	√	Faulty D *(CUT)*
8	√	√	X	X	Faulty A *(CUT)*
9	X	√	√	X	Faulty B *(CUT)*
10	X	√	X	√	Fault not in PLB
11	√	X	√	X	Fault not in PLB
12	X	X	X	√	Faulty D
13	√	X	X	X	Faulty A
14	X	X	√	X	Faulty C
15	X	√	X	X	Faulty B
16	X	X	X	X	Fault not in PLB

of inputs of a LUT is no more than the number of PLB outputs emanating from FFs; this is the case with most current FPGAs like those from Xilinx.

Table I shows all possible 2^4 gross syndromes for the four sessions and the inferences drawn from them. If a single PLB is faulty it should have a "X" in the two sessions in which it is configured as a CUT under the assumption of at most one faulty PLB.

Theorem 2: BISTer-1 is 1-diagnosable.

Proof: The theorem is proved by construction. We classify all the 16 outputs (rows in Table I) into six groups (cases). The same diagnostics apply to all rows of a given case.

Case 1: (Row 1 of the table). All the 4 sessions report *pass*. If a single PLB is faulty then there should be a *fail* when it is configured as a CUT. Since there is no *fail* in any session, no PLB is faulty.

Case 2: (Rows 2-5). In this case, only one session reports *fail* and since a PLB is configured as CUT twice so there should at least be two failing sessions. Hence the fault is not in a PLB. The fault could be in the interconnects or it could be a transient fault.

Case 3: (Rows 6-9). In this case, the two failing sessions are consecutive. Thus the two consecutive failing sessions identifies the faulty PLB as the one that is a CUT in these two sessions—in BISTer-1 a unique PLB is configured as CUT in two consecutive sessions. For example, for row 6, gross syndromes of sessions S_1 and S_2 report *fail*. PLB *C* is configured as a CUT in these two sessions, and hence the faulty PLB is *C*. Similar reasonings holds for rows 7-9.

Case 4: (Row 10-11). In this case, the two failing sessions are alternate. Since a PLB is configured as a CUT in two consecutive sessions and not in alternate sessions, and the PLB should *fail* at least when it is a CUT, so for this case no PLB is faulty. Again, the fault maybe in an interconnect or may be transient.

Case 5: (Rows 12-15). There are three failing sessions. The gross syndromes report *fail* when a PLB is configured as a CUT (two sessions) and when it is an ORA. Whenever the faulty PLB is configured as a TPG the gross syndrome is a

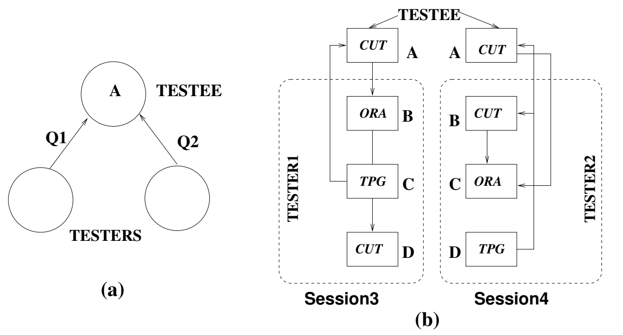


Fig. 7. (a) Testing graph for BISTer-1. (b) Tester configurations for PLB *A*.

pass ("√"), since, even if the TPG exercises the fault in the PLB, identical test vectors are fed to the two fault-free CUTs and compared by the fault-free ORA. Hence for row 12, the faulty PLB is *D*, since *D* is configured as a TPG in session S_4 whose gross syndrome is a *pass*. Similar analysis holds for rows 13, 14 and 15.

Case 6: (Row 16). In this case, all the sessions report *fail*. This case is not possible with at most one faulty PLB, since when the faulty PLB is configured as a TPG, the gross syndrome should be a *pass*.

We thus see that for each faulty PLB, the gross syndrome is unique. Hence BISTer-1 is 1-diagnosable. ⋄

We next show that BISTer-1 is not 2-diagnosable.

Theorem 3: BISTer-1 is not 2-diagnosable.

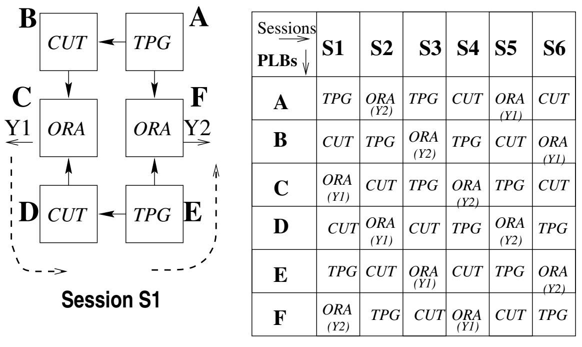
Proof: Figure 7a shows the "testing graph" for a PLB in BISTer-1. The *testing graph* of BISTer-1 has directed arcs between each tester-testee pair. Since each PLB is a *testee* the two times it is configured as a CUT, and the rest of the PLBs, in two different configurations, form the two *testers*, each PLB has two incoming arcs.

The testing graph fits the PMC fault-diagnosis model [15] which states that an upper bound on the diagnosability of a system is one less than the minimum in-degree of a node. Thus BISTer-1's diagnosability is at most one. The main argument of the PMC model (applied here) is that when syndromes $Q1 \neq Q2$, it is not possible to distinguish between the two fault patterns (each with 2 faults): 1) one of the testers (i.e., a non-TPG PLB in it) and *A* are faulty, and 2) both testers (non-TPG PLBs in them) are faulty (e.g., *D* in Tester1 and *C* in Tester 2) and *A* is fault-free. ⋄

B. A 2-diagnosable BISTer

The BISTer-2 architecture which has six PLBs is shown in Fig. 8a. Two PLBs are configured as TPGs, two as CUTs, and two as ORAs. Y_1 and Y_2 are the outputs at the first and second ORA respectively. The first ORA compares the outputs of the two CUTs and the second ORA compares the outputs of the two TPGs. Since there are six PLBs, there are six sessions; Fig. 8b shows the PLB configuration for each session. The gross syndromes corresponding to Y_1, Y_2 are denoted also by Y_1, Y_2, while the corresponding detailed syndromes are denoted by dY_1, dY_2, respectively. Furthermore, the joint dY_1, dY_2 syndromes for session S_i is denoted simply by dS_i.

In the testing graph of BISTer-2, each testee PLB has four incoming arcs, since it is twice tested as a CUT and twice



Sessions → / PLBs ↓	S1	S2	S3	S4	S5	S6
A	TPG	ORA (Y2)	TPG	CUT	ORA (Y1)	CUT
B	CUT	TPG	ORA (Y2)	TPG	CUT	ORA (Y1)
C	ORA (Y1)	CUT	TPG	ORA (Y2)	TPG	CUT
D	CUT	ORA (Y1)	CUT	TPG	ORA (Y2)	TPG
E	TPG	CUT	ORA (Y1)	CUT	TPG	ORA (Y2)
F	ORA (Y2)	TPG	CUT	ORA (Y1)	CUT	TPG

(a) (b)

Fig. 8. (a) BISTer-2 architecture. (b) Six sessions of BISTer-2.

TABLE II

GROSS Y_1, Y_2 SYNDROMES FOR BISTER-2 FOR ONE FAULTY PLB. WHEN A FAULTY PLB IS EITHER A TPG OR ORA ITS SYNDROME IS INDICATED BY A $\star$ WHICH MEANS THAT THE SYNDROME CAN EITHER BE A X OR $\surd$ DEPENDING ON WHETHER ITS FAULT(S) ARE EXERCISED OR NOT, RESPECTIVELY, BY ITS CURRENT FUNCTIONALITY (TPG OR ORA).

Faulty PLB		S_1	S_2	S_3	S_4	S_5	S_6
A	Y_1	$\star$	$\surd$	$\star$	X	$\star$	$\star$
	Y_2	$\star$	$\star$	$\star$	$\surd$	$\surd$	$\surd$
B	Y_1	X	$\star$	$\surd$	$\star$	X	$\star$
	Y_2	$\surd$	$\star$	$\star$	$\star$	$\surd$	$\surd$
C	Y_1	$\star$	X	$\star$	$\surd$	$\star$	X
	Y_2	$\surd$	$\surd$	$\star$	$\star$	$\star$	$\surd$
D	Y_1	X	$\star$	X	$\star$	$\surd$	$\star$
	Y_2	$\surd$	$\surd$	$\surd$	$\star$	$\star$	$\star$
E	Y_1	$\star$	X	$\star$	X	$\star$	$\surd$
	Y_2	$\star$	$\surd$	$\surd$	$\surd$	$\star$	$\star$
F	Y_1	$\surd$	$\star$	X	$\star$	$\star$	$\star$
	Y_2	$\star$	$\star$	$\surd$	$\surd$	$\surd$	$\star$

as a TPG. Thus according to the PMC model [15] is at most 3-diagnosable. We next establish its diagnosability.

Theorem 4: Assuming that (1) there is no fault masking of **all** detailed syndromes in the presence of two faults, and (2) in a BISTER-2 tile, faulty PLBs either uniformly all fail as TPGs and ORAs or uniformly all pass as TPGs and ORAs, BISTer-2 is 2-diagnosable with very high probability.

Proof: We first show that BISTer-2 is 1-diagnosable, irrespective of whether a faulty PLB fails as a TPG or as an ORA. Table II which is self-explanatory shows that the gross syndromes Y_1, Y_2 for each faulty PLB is unique; in particular note the unique pattern of three consecutive $\surd$ (*pass*) syndromes that occur in Y_2 for each faulty PLB (these three consecutive $\surd$'s start at a different session for each faulty PLB). E.g., for faulty PLB A, Y_2 is uniquely $\surd$'s in sessions S_4, S_5, and S_6. Hence BISTer-2 is 1-diagnosable.

We now show that BISTer-2 is 2-diagnosable for the case that the two faulty PLBs also fail as TPGs and ORAs; the proof for the case that they pass as TPGs and ORAs is similar. We first note that when a CUT is tested as an ORA (the CUT is exhaustively tested), it is easy to detect the case when its output is stuck-at-0. If that is the case, then when this PLB is an actual ORA the gross syndrome corresponding to its output is taken as a *fail* instead of the *pass* syndrome indicated by its stuck-at-0 state. With this scenario, the assumption that fault masking does not occur for *all detailed syndromes* is a very high probability one. Table III shows the gross syndromes Y_1 and Y_2 for different faulty PLB pairs. From the six session columns we see that Y_1 and Y_2 for each faulty pair is unique except for faulty pairs *AD*, *BE* and *CF*. For these pairs, Y_1, Y_2 are *fail*s in all sessions. We thus need additional analysis via the detailed syndromes to diagnose these pairs. For faulty pair *AD*, PLB A is configured as a TPG in session S_1 and S_3, while PLB D is configured as a CUT in these sessions; see Fig. 8b. The detailed syndromes dY_1 and dY_2 for sessions S_1 and S_3 will be thus be identical—each faulty PLB is configured to perform the same function in both sessions. Also for sessions S_4 and S_6, A is a CUT and D is a TPG. Hence S_4 and S_6 will also have identical detailed syndromes Using the same reasoning for faulty pairs *BE* and *CF* we have: For *AD*: $dS_1 = dS_3$, $dS_4 = dS_6$; For *BE*: $dS_1 = dS_5$, $dS_2 = dS_4$; For *CF*: $dS_2 = dS_6$, $dS_3 = dS_5$.

We may not be able to distinguish faulty pairs *AD* and *BE* when $dS_1 = dS_3 = dS_5$ and $dS_2 = dS_4 = dS_6$. However, this is a very unlikely event. For example, consider *AD* as the faulty pair. In session S_3, A is a TPG and D is a CUT, while in S_5, A and D are ORAs. The dY_1 syndrome for S_3 gives the results of testing D as a CUT using non-faulty PLBs (E as the ORA and F as the other CUT), while dY_1 for S_5 is the result of testing two non-faulty CUT's (B, F) using a faulty PLB A as the ORA. One can see that it is very unlikely that the dY_1 0/1 bits in S_3 will match the dY_1 0/1 bits in S_5, since for that to happen D needs to fail as a CUT in S_3 for exactly those input test patterns for which A fails as an ORA in S_5. Similarly, dY_2 of S_3 and S_5 will only match if A fails in S_3 as a TPG for exactly those test patterns for which D fails in S_5 as an ORA–a very low probability event. Thus when *AD* is the faulty pair, $dS_3 = dS_5$ only if two very low probability events occur. Similarly, $dS_4 = dS_6$ if two very low probability events occur. Thus *AD* and *BE* faulty pairs will be indistinguishable, only if four very low probability events occur, making this situation astronomically unlikely. Similarly, any detailed syndrome equality between any of the other three faulty pairs is extremely unlikely.

Finally, all the gross syndromes (Y_1, Y_2) of Table II (1 faulty PLB) are distinct from those of Table III (2 faulty PLBs); note again the unique pattern of three consecutive $\surd$ (*pass*) syndromes in Table II that occur in Y_2 for each faulty PLB, and which do not occur in Y_2 for any of the entries in Table III. We can thus distinguish between the syndromes for single and double faults. $\diamond$

V. FUNCTIONAL TESTING AND DIAGNOSIS

We present here a functional testing and diagnosis (TAD) technique Fast-TAD, that, in conjunction with our BISTer designs of the previous sections, detects failures of PLBs only in two possible functional modes they will be used in as the ROTE moves across the FPGA in either a fault-free scenario or in the presence of reconfigured faults. As mentioned earlier, functional testing is possible only when it is known which circuit(s) will be mapped to the FPGA. For simplicity of exposition, we assume that only one circuit will be mapped to the FPGA; the extension to multiple circuits is straightforward.

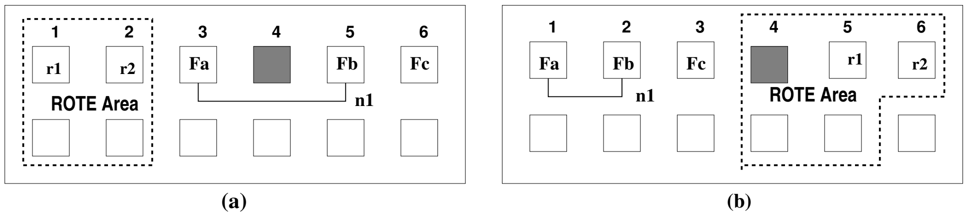


Fig. 9. ROTE movement in the presence of a single f-faulty PLB shown dark. F_i represents the functionality of each PLB. (a) Initial position of the ROTE. PLB 3 implements its *original function* F_a; PLB 6 is 2 fault-free PLBs to its right. (b) ROTE occupies columns 4 and 5. Note the warped shape of the BISTer-1 tile needed to occupy four fault-free PLBs. PLB 6 is thus occupied causing its original function F_c to be mapped to PLB 3.

TABLE III
GROSS Y_1, Y_2 SYNDROMES FOR BISTER-2 IN THE PRESENCE OF TWO FAULTY PLBS, ASSUMING THAT A FAULTY PLB ALSO FAILS AS A TPG AND AS AN ORA AND THAT FAULT MASKING DOES NOT OCCUR FOR ALL DETAILED SYNDROMES (A VERY HIGH PROBABILITY ASSUMPTION).

Faulty PLBs		S_1	S_2	S_3	S_4	S_5	S_6
AB	Y_1	X	X	X	X	X	X
	Y_2	X	X	X	X	✓	✓
AC	Y_1	X	X	X	X	X	X
	Y_2	X	X	X	X	X	✓
AD	Y_1	X	X	X	X	X	X
	Y_2	X	X	X	X	X	X
AE	Y_1	X	X	X	X	X	X
	Y_2	X	X	X	✓	X	X
AF	Y_1	X	X	X	X	X	X
	Y_2	X	X	X	✓	✓	X
BC	Y_1	X	X	X	X	X	X
	Y_2	✓	X	X	X	X	✓
BD	Y_1	X	X	X	X	X	X
	Y_2	✓	X	X	X	X	X
BE	Y_1	X	X	X	X	X	X
	Y_2	X	X	X	X	X	X
BF	Y_1	X	X	X	X	X	X
	Y_2	X	X	X	X	✓	X
CD	Y_1	X	X	X	X	X	X
	Y_2	✓	✓	X	X	X	X
CE	Y_1	X	X	X	X	X	X
	Y_2	X	✓	X	X	X	X
CF	Y_1	X	X	X	X	X	X
	Y_2	X	X	X	X	X	X
DE	Y_1	X	X	X	X	X	X
	Y_2	X	✓	✓	X	X	X
DF	Y_1	X	X	X	X	X	X
	Y_2	X	X	✓	X	X	X
EF	Y_1	X	X	X	X	X	X
	Y_2	X	X	✓	✓	X	X

Clearly, Fast-TAD should be much faster than the exhaustive TAD methods of previous work. In the rest of this section, we present the Fast-TAD method in conjunction with BISTer-1; a similar approach can be used for functional testing with BISTer-2.

Functional testing requires considerations of issues beyond just restricting the function configured into a CUT to be its function in the "normal" circuit (i.e., when there is no roving tester). Issues that we address here are:

1. As the roving tester ROTE moves across the FPGA, and the application circuit is configured to accommodate this, each PLB can be configured with different circuit functions; these are called its *operational functions*. Is there a limited number of operational functions that a PLB will be configured with, for any reconfigured fault pattern (including the fault-free pattern), as the ROTE moves across the FPGA, and is it possible to know these a-priori before the ROTE starts roving?
2. Assume that the answer to the above question is in the affirmative, and we know that each PLB will be configured with k different and known circuit functions as the ROTE moves across the FPGA. Then, when a PLB A is in the ROTE area (i.e., in a BISTer-1 tile), it must be ensured that all k configurations that occur in it during ROTE's movement are tested. In a straightforward method, this means that A will need to be configured with a total of $4k$ functions in the two sessions in which it is a CUT—$2k$ functions each time it is a CUT, e.g., when PLBs A, D are CUTs in a session, they would each be configured with the k operational functions of A as well as the k such functions of D in a straightforward application of functional testing. The question, however, is whether it is possible to reduce the total number of functional configurations for the CUT A (and thereby reduce the test time) below $4k$, while satisfying the requirement of testing each CUT for all its k operational functions (across all its test sessions).

These issues and the diagnosability of BISTer-1 using functional testing are tackled in the rest of this section.

A PLB X is said to be *functionally-faulty (f-faulty)* if fault(s) in X cause incorrect output(s) to be produced for one or more input vectors when X implements any of its operational functions. Fast-TAD only detects and diagnoses f-faulty PLBs; faulty PLBs that are not f-faulty are accurately deemed to be "good" as they do not affect the correct functioning of the circuit.

The first question posed above is answered in the following lemma.

Lemma 1: While roving the ROTE left to right in an FPGA either without f-faults or with reconfigured f-faults, a PLB needs to implement at most two functions, its original function (determined when the ROTE is in its initial left-most position) and the function of the PLB two f-fault-free PLBs to its right in the same row.

Proof Sketch: A special case of this lemma is illustrated in Fig. 9 in which the ROTE moves across the FPGA in the presence of a single f-faulty PLB. Note that each PLB implements at most two functions specified in the lemma, as shown explicitly for PLB 3. ◇

Following Lemma 1, for a PLB X, we denote x_1, the *original function* of X, as the circuit function it implements when the ROTE is in its initial (leftmost two columns position, and x_2 as the circuit function mapped to it when the ROTE's leftmost column is to the immediate right of X's column (in

Sessions → PLB ↓	S1	S2	S3	S4
A	TPG	ORA	d1,d2 CUT	a1,a2 CUT
B	b1,b2 CUT	TPG	ORA	a1,a2 CUT
C	b1,b2 CUT	c1,c2 CUT	TPG	ORA
D	ORA	c1,c2 CUT	d1,d2 CUT	TPG

(a)

Sessions → PLB ↓	S1	S2
A	TPG	b1,b2 CUT
B	c1,c2 CUT	b1,b2 CUT
C	c1,c2 CUT	ORA
D	ORA	TPG

(b)

Sessions → F-faulty PLB ↓	S1	S2	S3	S4
A	✓	X/✓	X/✓	X
B	X	✓	X/✓	X/✓
C	X/✓	X	✓	X/✓
D	X/✓	X/✓	X	✓

(c)

Sessions → F-faulty PLB ↓	S1	S2
A	✓	X/✓
B	X/✓	X
C	X	X/✓
D	X/✓	✓

(d)

Fig. 10. (a)-(b) PLB configuration for the first and second test sets, respectively, with the functions tested in a CUT in each session. (c)-(d) Gross syndromes for each test set.

other words, x_2 is the original function of the PLB two f-fault-free PLBs to X's right in the same row). In Fig. 9, for PLB 3, for example, $x_1 = F_a$ and $x_2 = F_b$. x_1 and x_2 are the only two operational functions/configurations of any PLB X in any reconfigured fault situation.

We now address the second question posed at the beginning of this section. Since $k = 2$, as established above, a straightforward application of functional testing to BISTer-1 would require each PLB A in the BISTer to be configured with four functions each time it is a CUT (e.g., when A, D are CUTs in session S_3 [see Fig. 10a], both A and D would be configured with functions a_1, a_2, d_1, d_2), resulting in eight total functional configurations of each CUT across all four sessions. It is, however, possible to reduce the total number of configurations of each CUT to almost half this number as we discuss below.

Fast-TAD uses the BISTer-1 architecture of Sec. IV, but uses two sets of tests. The first test set, uses all four BISTer-1 sessions, while in the second test set, which is used for further diagnosis, only one session is adaptively used. In the first test set, each PLB X is a CUT twice. In one of its CUT sessions, it is tested with configurations x_1 and x_2, while in its second CUT session, it is tested with configurations y_1 and y_2, where Y is the other CUT in that session. Figure 10a shows all the sessions in the first test set, along with the functions configured in the CUTs. Note that for each CUT, its two operational functions are tested in exactly one session. Thus all operational functions of all PLBs in the BISTer are covered in the first test set. Figure 10c shows the gross syndrome for PLB configurations in Fig. 10a. When the f-faulty PLB is configured as a TPG then the gross syndrome is a *pass*. When it is configured as a CUT and implements its operational functions, then the gross syndrome is a *fail* . In all other cases it is either a *fail* or a *pass*.

The second test set (Fig. 10b) is used only to distinguish between the possible f-fault being in either of A, C or in either of B, D (each PLB in the two pairs have common gross syndromes see Fig. 10c). Only one further session of BISTer-1 is needed to distinguish between the above PLBs. As shown in Fig. 10d, session S_1 is needed to distinguish between either of A, C being f-faulty, while session S_2 is needed to distinguish between either of B, D being f-faulty. Note that this second test is needed only if a syndrome common to either of the above pairs occurs. The diagnostics are explained further in the proof of Theorem 5.

Theorem 5: The Fast-TAD method using BISTer-1 can diagnose one f-faulty PLB in each BISTer-1 tile.

Proof: As shown in Fig. 10c, the gross syndrome vector (*pass/fail*) for all four sessions, when PLB A is f-faulty are disjoint from those of PLBs B and D. Also the gross syndrome vectors for f-faulty PLB D are disjoint from those of f-faulty PLBs A and C. However, as shown in Fig. 10c, we may not be able to distinguish between f-faulty PLBs A, C and between f-faulty PLBs B, D. The second test (Fig. 10b) is performed in case a gross syndrome that is common to the f-fault being in either A, C occurs (in this case only session S_1 of Fig. 10b is performed), or a gross syndrome that is common to the f-fault being in either B, D occurs (in this case only session S_2 of Fig. 10b is performed). We see from Fig. 10d that PLBs *A, C* have different gross syndromes in session S_1, and that PLBs *B, D* have different gross syndromes in session S_2. Hence using the two set of tests, and only one session in the second test (if needed), we get unique gross syndromes for each f-faulty PLB. ⋄

Hence in FAST-TAD using BISTer-1, in the fault-free case, each CUT will be configured with a total of only four functions over all sessions (test set 2 will not be required in this case). When there are one or more faults, only two of the 15 remaining gross syndromes lead to the second test set with exactly one session (S_1 or S_2 in Fig. 10(b) depending on the gross syndrome) in which the one pair of CUTs tested is configured with two functions. Thus if p is the fault probability of a PLB, the average number of functional configurations required for each PLB per ROTE movement across the FPGA is $4 + 2 \times \frac{2}{15}(1 - (1 - p)^4)$. E.g., for p=0.01, this number is 4.01, and for $p = 0.1$, it is 4.09, as opposed to always 8 for the straightforward method. Thus almost a factor of two improvement in test time is obtained over the latter by using our novel functional testing technique.

Finally, in the broader context of fault tolerance mentioned in Sec. II, where spare PLBs need to be configured into the circuit, we briefly discuss the issue of functional testing for these PLBs. We propose that in order to reduce test time these PLBs be tested only when they need to be configured into the circuit. The fault reconfiguration method (e.g., [14]) can identify the spare PLBs that will be configured in, which PLBs they will replace, and hence what their operational functions will be. We can then perform FAST-TAD functional testing for the spare PLBs by including them in BISter-1 tiles. If any spare PLB fails such a test (is diagnosed as faulty), then the reconfiguration algorithm can identify another spare PLB as a replacement, and so forth, until we identify the required functionally-correct spare PLB. On the issue of accumulation of faults in untested spare PLBs until they are needed, note that these PLBs are not connected to any track segment and are thus effectively isolated from the rest of the system. Of course, an interconnect BIST method (e.g., [18], [19]) should periodically check that the switches in the PLB-to-track interconnection structure are all set to the off or disconnect state. Also, due to the large number of interconnect tracks in a routing channel in current FPGAs, adjacent PLBs are sufficiently separated from each other that dormant faults in them should not affect the other. Thus faults that may accumulate in spare PLBs or in circuit PLBs that do not affect their operational functions, have little likelihood of affecting correct circuit operation.

VI. Applications to Current FPGAs

The maximum length m of a test vector generated by the TPG should be equal to the number of inputs to the CUT. Hence a TPG must have m output pins to test an m-input PLB when it is a CUT. Thus if the number of output pins of a PLB is q, then the number of PLBs required for the TPG is $t = m/q$. For current FPGAs, $t \approx 3$, and thus three PLBs are required in a TPG for generating test vectors for the entire PLB. However, because of the multiple-block structure of current SRAM based FPGAs (e.g., Xilinx Virtex-II), explained in more detail below, it is possible to use TPGs with fewer PLBs, and in particular, a single-PLB TPG, for sequentially testing smaller parts of a PLB (e.g., logic units, look-up tables) using smaller-length test vectors, thereby testing the entire PLB.

In the following, we provide techniques for:

1. A 1-PLB TPG to sequentially test a CUT PLB in the Xilinx Virtex-II FPGA (a similar approach can be used for other current FPGAs as well) that can be used in BISTer-1 in a straightforward way.
2. A 3-PLB TPG used in BISTer-1 to test all parts of a PLB simultaneously (useful for FPGAs with $t \approx 3$, which seems to be the typical value for this ratio in current FPGAs). With such a TPG, BISTer-1 becomes a 2×3 PLB tile and some modifications are needed in its operation compared to the 1-PLB TPG BISTer-1. These are explained below and theoretical results proven for the diagnosability of this modified BISTer-1; we refer to this BISTer-1 configuration as *BISTer-1*$^{2\times3}$.

A. BISTer-1 with a 1-PLB TPG for Current FPGAs

Present technology FPGAs like Xilinx Virtex-II have PLBs that are formed of multiple building blocks called *Logic Cells (LC)*. Each LC contains a function generator, i.e., a look-up table (LUT), carry and control logic and a storage element (flip-flop [FF]). Each of these i-input LCs can be configured independently to perform any i-input boolean function. Two LCs can also be multiplexed to perform a j-input boolean function ($j > i$), with the control signal of the multiplexor coming from a separate PLB input. Fig. 11 shows the schematic of the PLB with multiple LCs. Also, in most current FPGAs, q the number of *flip-flopped (FF'ed) outputs* (output pins that are outputs of flip-flops in the PLB) of the PLB, is greater than i, the number of inputs to each LC. Note that the outputs of a TPG are all essentially state bits of finite-state machine, and thus will all need to be flip-flop outputs. Hence, even though the total number of PLB outputs may be greater than q, for the purpose of TPG design, we can only consider that subset of PLB outputs that are FF'ed outputs. Most current Xilinx FPGAs, for example, like the Virtex-II, Virtex-II Pro, Virtex-4, Spartan 3E, Spartan 3/3L, have $i = 4$ and $q = 8$ FF'ed outputs along with other non-FF'ed outputs. Thus we can apply the technique described below, which requires that $q > i$, to use a 1-PLB TPG to sequentially generate test vectors for all LCs in a CUT PLB.

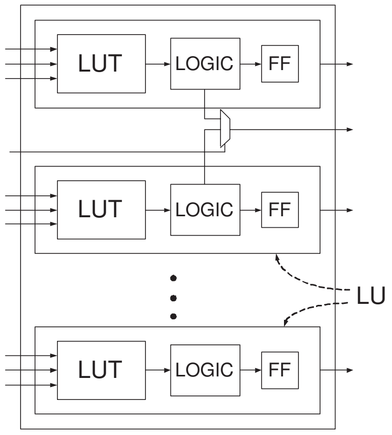


Fig. 11. Schematic of a PLB formed of multiple Logic Cells (LCs).

With a TPG formed of a single PLB with the design given in Fig. 11, the following procedure can then be employed in each testing session in BISTer-1:

1. The TPG PLB with q ($\geq i+1$) outputs produces i-bit test vectors, and an additional bit, if needed (see below), that is held constant at a 0 or a 1 depending on which of two possible multiplexed LCs is being currently tested.
2. If the LCs of the CUT are independent (i.e., the LCs are not multiplexed) then the i-bit test vectors generated by the single PLB TPG are simultaneously passed to each i-input LC of the CUT. The output of each LC of the CUT is then compared with the output of the corresponding LC of the other CUT PLB.
3. If two LCs of the CUT are multiplexed then only one LC is tested at a time by passing the i-bit test vector to it and comparing it with the corresponding LC of the other CUT. As shown in Fig. 12a the upper LC of the CUT PLB, enabled by keeping the control signal of the multiplexor at a constant value 0, is under test. After the testing of the first LC in the multiplexed structure is completed, the second LC in this structure is enabled by changing the control signal value of multiplexor and is tested in a similar fashion by passing i-bit test vectors to its input (see Fig. 12b).

B. *BISTer-1*$^{2\times3}$ *with a 3-PLB TPG for Current FPGAs*

BISTer-1$^{2\times3}$ is similar to the 2×2 BISTer-1 of Sec. IV-A except that there are two extra PLBs (in the third column) acting as the two extra TPG PLBs, as shown in Fig. 13. There

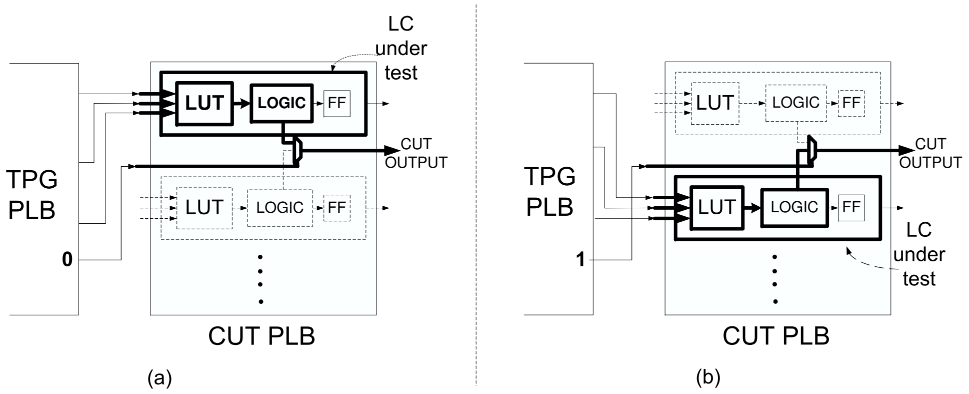


Fig. 12. Testing of multiplexed LCs in a CUT. (a) The upper LC of the multiplexed structure is tested by passing test vectors from the TPG to its input with the MUX control input held at 0. (b) The lower LC in the multiplexed structure is under test; the the MUX control input is constant at 1.

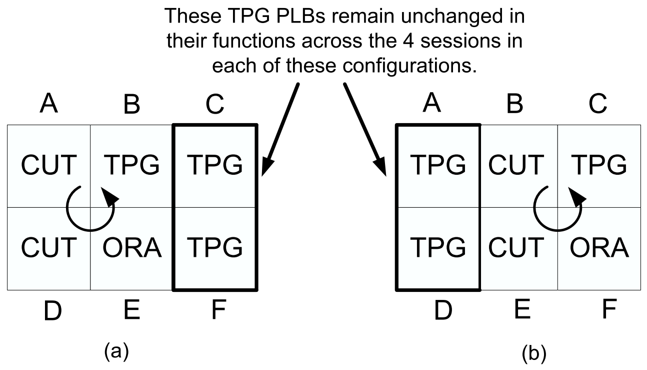

These TPG PLBs remain unchanged in their functions across the 4 sessions in each of these configurations.
A
B
C
CUT
TPG
TPG
CUT
ORA
TPG
D
E
F
(a)
TPG
CUT
TPG
TPG
CUT
ORA
(b)

Fig. 13. BISTER-$1^{2\times3}$ tile: (a) In the first configuration, PLBs A, B, D and E are tested in a 2×2 BISTer-1 fashion in 4 sessions, while PLBs C and F function as the two additional TPG PLBs. (b) The second configuration in which PLBs D, E, C, F are tested in 4 session and PLBs A and B are the two additional TPG PLBs.

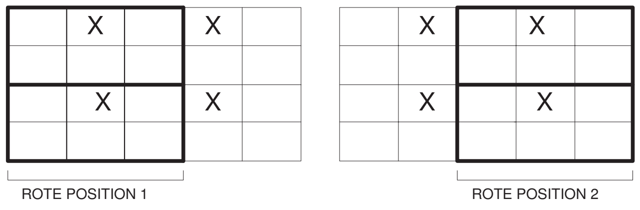


Fig. 14. Faults in alternate columns and rows diagnosed by BISTer-$1^{2\times3}$. The areas inside the dark lines are the BISTer areas in the corresponding ROTE positions.

are two configurations, each with four sessions in this BISTer. In the first configuration with the first four sessions, PLBs A, B, D and E are tested in a 2×2 BISTer-1 fashion with PLBs C and F functioning as the two extra TPG PLBs. In the next configuration with the rest of the four sessions, PLBs B, C, E and F are tested in a 2×2 BISTer-1 fashion with PLBs A and D functioning as the extra TPG PLBs. Once these eight sessions are completed, the BISTer tile moves one row down and repeats another set of eight sessions. In BISTer-$1^{2\times3}$, the ROTE moves by two columns instead of one that it did for the 2×2 BISTer-1. This is because, if the ROTE is moved by one column instead of two, then testing of the sub-area B, C, E and F will be unnecessarily repeated.

a) Diagnosabilities of BISTer-$1^{2\times3}$:: Note that BISTer-$1^{2\times3}$ can be used in the exhaustive testing mode (as in the BISTer-1 of Sec. IV-A) or in the functional mode as described in Sec V.

Theorem 6: BISTer-$1^{2\times3}$'s diagnosabilities in either the exhaustive or functional PLB testing modes are as follows.

(a) It can diagnose one faulty (f-faulty) PLB in each 2×3 sub-area in the exhaustive (functional—Fast-TAD) testing mode.

(b) It can diagnose a faulty (f-faulty) PLB in every alternate column or row in the exhaustive (functional—Fast-TAD) testing mode.

(c) It can diagnose two faulty (f-faulty) PLBs in consecutive rows in the exhaustive (functional—Fast-TAD) testing mode.

Proof: (a) Considering one fault in a 2×3 BISTer sub-area, if the fault is in either of A, B, D or E then for exhaustive testing, from Theorem 2, it will be diagnosed in the first four sessions, as the extra TPG PLBs C and F are fault- free. Similarly, if the fault is in C or F, then it will be diagnosed in the last four sessions where the extra two TPG PLBs will be fault-free PLBs A and D. A similar conclusion follows from Theorem 5 when BISTer-$1^{2\times3}$ is used in the functional testing mode (Fast-TAD).

(b) If the faults (or f-faults) are in alternate columns or rows, then each fault will fall in a different 2×3 BISTer area where it will be the only fault in that particular sub-area. This is shown in Fig. 14. As there will be only one fault per BISTer area and since BISTer-$1^{2\times3}$ is 1-diagnosable from part (a) of this theorem, all the faults in alternate columns or rows will be diagnosed.

(c) As BISTer-$1^{2\times3}$ moves one column down after eight sessions, hence the two consecutive faults (or f-faults) will fall in two different BISTer sub-areas as shown by Fig. 15. Thus from part (a) above, each faulty or f-faulty PLB will be diagnosed. ⋄

The following corollary provides more analysis of the fault patterns of at most one fault in disjoint 2×2 subarrays (at most one PLB fault out of 4 PLBS) that BISTer-$1^{2\times3}$ can diagnose; note that BISTer-1 can diagnose all patterns with at most one

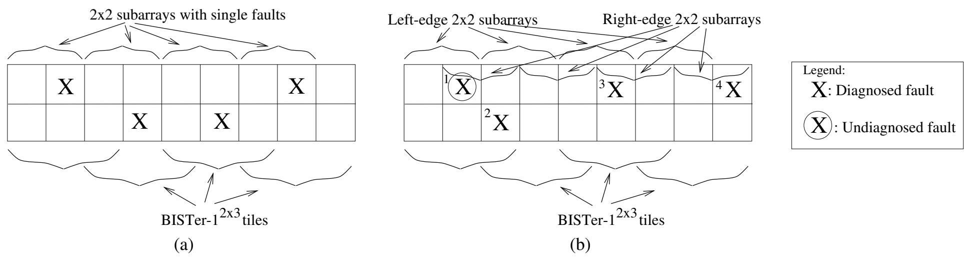


Fig. 16. Fault patterns for Corollary 1. (a) A pattern of single faults, shown by X's, in disjoint 2×2 subarrays that is completely diagnosable by BISTer-1$^{2\times3}$. (b) Another pattern of faults in disjoint 2×2 subarrays that is partially (three of four) diagnosable by BISTer-1$^{2\times3}$—fault 2 meets condition (i), fault 3 meets conditions (i) and (ii) (both the left-edge and right-edge 2×2 subarrays that it lies in meet their respective conditions) and fault 4 meets condition (ii) of Corollary 1.

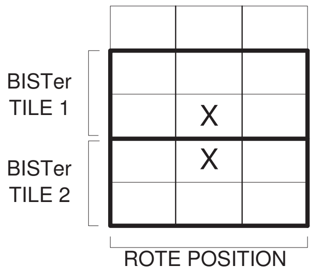


Fig. 15. Two consecutive faults in rows are diagnosed by BISTer-1$^{2\times3}$. Areas inside the dark lines are the BISTer areas.

fault in disjoint 2×2 subarray (Theorem 2).

Corollary 1: Assuming that the ROTE moves by two columns to its next position, BISTer-1$^{2\times3}$ can correctly diagnose a single fault in any 2×2 subarray that is within any region tested by it, i.e., any region made up of the 2×3 tiles occupied and tested by BISTer-1$^{2\times3}$ if one of the following conditions is met:
(i) If a 2×2 subarray A is at the **left edge** of a 2×3 tile occupied by BISTer-1$^{2\times3}$, then any single fault in A can be correctly diagnosed if there is no fault in the left column of the 2×2 subarray B that is to the immediate right of A.
(ii) If a 2×2 subarray A is at the **right edge** of a 2×3 tile occupied by BISTer-1$^{2\times3}$, then any single fault in A can be correctly diagnosed if there is no fault in the right column of the 2×2 subarray C that is to the immediate left of A.

This means that BISTer-1$^{2\times3}$ can *completely diagnose* (i.e., diagnose all the faults correctly) for at least 50% of all fault patterns with at most a single fault in disjoint 2×2 subarrays of the FPGA; see Fig. 16(a) for an example of such a fault pattern that is completely diagnosable by BISTer-1$^{2\times3}$. For other fault patterns in the above category that BISTer-1$^{2\times3}$ is unable to completely diagnose, it may be able to diagnose them partially, i.e., diagnose some of their faults but not all (see Fig. 16(b)).

Proof: Consider any 2×2 subarray A in a 2×3 tile occupied by BISTer-1$^{2\times3}$. If A is at the left edge of BISTer-1$^{2\times3}$ (e.g., the "left-edge" subarrays in Fig. 16(b) in which faults 1 and 2 lie), then condition (i) of the corollary means that the 2×1 column to the immediate right of A is non-faulty. This implies that there is exactly one fault in this 2×3 tile and from Theorem 6 this fault will be diagnosed. Similarly, if A is at the right edge of a BISTer-1$^{2\times3}$ tile (e.g., the "right-edge" subarray in Fig. 16(b) in which fault 4 lies), then condition (ii) of the corollary means that there is exactly one fault in this 2×3 tile and from Theorem 6 this fault will be diagnosed.

If A is at the left edge of BISTer-1$^{2\times3}$, then, given that the 2×2 subarray B to its immediate left has a single fault in it, the probability that this fault is not in the left column of B [condition (i)] is 0.5, and if B has no faults in it, then condition (i) has a probability of 1. A similar analysis holds when A is at right edge of BISTer-1$^{2\times3}$. Thus BISTer-1$^{2\times3}$ will completely diagnose at least (exactly) 50% of all fault patterns with at most (exactly) a single fault in each disjoint 2×2 subarray of the FPGA. ⋄

Figure 16(a) shows a failure pattern that is completely diagnosable by BISTer-1$^{2\times3}$ while Figure 16(b) shows one that is partially diagnosable.

C. Pros and Cons of BISTer-1 and BISTer-1$^{2\times3}$

The disadvantage of BISTer-1 with a 1-PLB TPG in FPGAs with $t\approx3$ to the 3-PLB TPG BISTer-1 (BISTer-1$^{2\times3}$) is that it requires l separate configurations of the interconnections from the TPG to the PLB CUTs, and from the relevant CUT outputs to the ORA versus only one configuration for BISTer-1$^{2\times3}$; recall that t is the input/output pin ratio of a PLB and l is the number of logic cells (LCs) and hence LUTs in a PLB. This can appreciably increase the testing time for BISTer-1. This is ameliorated somewhat by the fact that since in BISTer-1 smaller parts of the CUT PLBs are tested at a time, fewer total test vectors are needed to test the entire PLB ($l\cdot2^{3q/l}$ versus 2^{3q} in BISTer-1$^{2\times3}$, where recall that q is the number of outputs and $3q$ the number of inputs of a PLB). Another ameliorating factor is that in the BISTer-1$^{2\times3}$ ROTE area, after the first round of testing (8 sessions per BISTer-1$^{2\times3}$ tile), each Bister tile (save the last) moves down by one row for a second round of Bister sessions. Thus the total number of sessions in each ROTE area in an $n\times m$ FPGA is $\approx 8n$ for BISTer-1$^{2\times3}$, while this number is $4\cdot(n/2)=2n$ for BISTer-1. However, since the BISTer-1 ROTE moves across the FPGA by one column to its next position, while the BISTer-1$^{2\times3}$ ROTE moves across the FPGA by two columns from one position to the next, the total test sessions for the entire FPGA for BISTer-1 is $2mn$ versus $\approx 8n(m/2)=4mn$ for BISTer-1$^{2\times3}$.

The 1-PLB-TPG BISTer-1, however, has the advantage of higher diagnosability (1 of 4 PLB faults [[Theorem 2]) compared to BISTer-1$^{2\times3}$ (1 of 6 PLB faults [Theorem 6(a)]). Note, however, from Theorem 6 and Corollary 1 that there are other fault patterns in the FPGA that the ROTE using BISTer-1$^{2\times3}$ can diagnose that alleviate this reduced diagnosability within the BISTer tile and results in high diagnostic coverage, as discussed in the next section.

D. BIST for FPGAs with PLBs with Larger Input/Output Pin Ratios

With rapid technology advances leading to higher integration and smaller feature sizes, it is likely that larger functionality will be packed in a PLB leading to increases in the PLB input/output pin ratio t. Our assumption of $t \approx 3$ is based on the ORCA 2C series FPGAs from Lucent Technologies; [1], [3], [4] use the same assumption also based on this family of FPGAs.

Some current FPGAs like those from Xilinx and Altera have PLB input/output pin ratios of around 4:1. However, these PLBs are basically composed of multiple slices. Recent FPGAs from Xilinx have four slices per PLB, and each slice has two LCs (logic cells). Each LC has two 4 input 1 output LUTs along with control logic and a flip-flop. Thus each slice has 8 inputs and two flip-flop outputs (along with other non-flipflop outputs). All these four slices in a PLB can be tested simultaneously by passing 8-bit test vectors to each slice. Even if two slices are combined to perform a single function (and at most two slices can be so combined), at most 16-input test vectors are required. Three PLBs, each having 8 outputs, are enough to produce 16-input test vectors. Thus BISTer-$1^{2\times3}$ can be used to test such a PLB in two rounds, one round per two slices of the PLB. In general, irrespective of the value of t, it should be possible to use either BISTer-1 or BISTer-$1^{2\times3}$ to test a PLB in a sequence of one or more rounds.

VII. Simulation Results

A 32×32 FPGA array with 3-input 1-output PLBs was functionally simulated in C with random functions mapped to each PLB—each LUT entry was randomly chosen to be 0 or 1 with a probability of 0.5. Test and diagnosis using two techniques was implemented on this FPGA: (1) Fast-TAD with BISTer-$1^{2\times3}$ in which as mentioned in Sec. VI in each ROTE position after each BISTer-$1^{2\times3}$ tile (except the bottom one) finishes testing it is shifted down by one row for another testing phase; as mentioned in Theorem 6 and Corollary 1, this BISTer is 1-diagnosable but can also provably diagnose many 2-fault patterns using non-adaptive diagnosis. (2) STAR-3×2 BISTer using the provably 1-diagnosable non-adaptive diagnosis technique of [1], [3]; see Secs. III-B and III-C. Finally note that STAR-3×2 BISTer performs exhaustive testing (it has no other mode of testing) [1], [3]. However, to reduce its fault latency below what would be normal for its mode of testing, we used the STAR-3×2 BISTer to test only 16 of the $2^{2^3} = 256$ possible functions of a 3-input LUT.

It is informative to note that while both BISTers use 6-PLB tiles, their structures and PLB functionality rotations for obtaining different test sessions are very dissimilar. In particular, STAR-3×2 BISTer rotates the PLB functionalities in a cycle to obtain its six sessions (see Sec. III-B and Fig. 3). On the other hand, in our BISTer-$1^{2\times3}$, the functionality rotations occur in two overlapped cycles in 2×2 subtiles (see Sec. VI-B and Fig. 13), each corresponding to the basic structure of the the 1-diagnosable (2×2)-tile BISTer-1 of Sec. IV-A, leading to 8 sessions of testing. As a result, in BISTer-$1^{2\times3}$ we are able to harness the 1-diagnosability of each of its two overlapped 2×2 subtiles leading to more patterns of provable diagnosabilities (beyond 1-diagnosability) as established in Theorem 6 and Corollary 1.

Three different types of PLB fault patterns were injected: (a) Randomly distributed faults with a given density. (b) Moderately clustered faults in which faults occur in a cluster around a center faulty PLB $\mathcal{C}$ with a probability distribution function (pdf) of $\frac{k}{(d+1)^2}$ where d is the Manhattan distance of a PLB from $\mathcal{C}$ and k is a suitable proportionality constant; each cluster is distributed randomly across the FPGA with a certain density (e.g., a 2% cluster density out of 1000 PLBs means 20 fault clusters are randomly distributed and each cluster will have multiple faults with the above pdf). (c) Strongly clustered faults in which faults occur in a cluster around a center fault with a pdf of $\frac{k}{d+1}$.

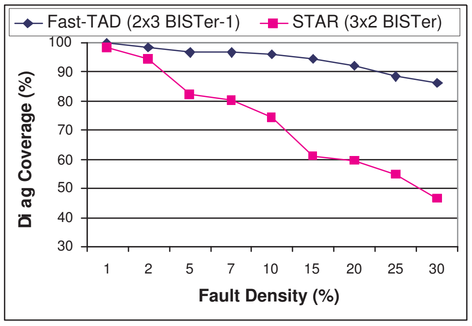
In both moderate and strong clusters, the probability of a PLB being faulty grows roughly linearly with the number of faults near it[6]. According to [12] this linear growth is the property that leads to the well-established Stapper yield model for chip defects [17]. Thus in our cluster model we are roughly approximating the clustering effect of defects that leads to the Stapper yield model.

We measured two metrics in our simulations of the two BISTers, *diagnostic coverage* defined as the percentage of faults that are correctly diagnosed, and *fault latency* defined as the average time from the occurrence of a fault to its correct diagnosis (in our simulations all faults are generated simultaneously at time 0). In the results, fault latencies are given in units of t_1, which is the time taken to test a PLB with only one configuration or function mapped to it[7].

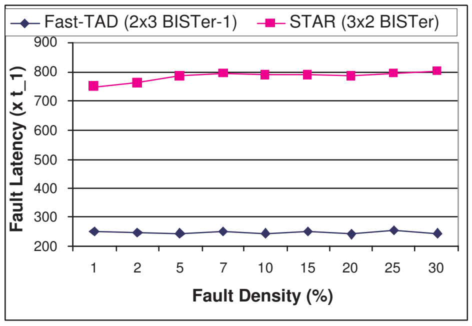
As we can see in Figs. 17-19 and Tables IV-VI, our Fast-TAD method using BISTer-$1^{2\times3}$ outperforms the STAR-3×2 BISTer in both diagnostic coverage and fault detection latency across different fault densities ranging from 1% to as high as 30%. For random faults, our technique is quite stable, giving a coverage of 96% at 10% fault density, while the STAR method's coverage falls rapidly to about 75% at this density. For clustered faults the absolute coverage gap between the two methods is about 30-38% (this represents a relative coverage gap of 50-68%), and the fault detection latency of the STAR method is about 3-4 times more than that of our technique. Thus our new methods significantly better the current state-of-the-art in on-line testing. This augurs well for the effective application of our techniques to current and emerging VDSM

[6]E.g., in the strong cluster model, if a PLB A is near two faulty PLBs X, Y, and at distances of d_x and d_y from them, respectively, then its fault probability is proportional to $\frac{1}{d_x+1} + \frac{1}{d_y+1}$.

[7]By definition, t_1 includes the *reconfiguration time* t_r to load in the new configuration bits to test a PLB and the actual *test time* t_t–time to generate test vectors and the corresponding syndromes at the output of the ORA–to test it; $t_1 = t_t + t_r$. Since our simulations are behavioral, we have kept $t_r = 0$ for both BISTers we compare. Note, however, that the latency comparisons are valid since, if in our simulations, the two BISTers take $q \cdot t_1 = q \cdot t_t$ and $r \cdot t_1 = r \cdot t_t$ times to test the entire FPGA under some fault density and distribution, then if a t_r time is included, the two BISTers will take $q(t_t+t_r)$ and $r(t_t+t_r)$ times to complete testing, giving the same percentage differences in their latencies.



(a)



(b)

Fig. 17. Results for random fault distribution: (a) Diagnostic coverage and (b) fault detection latency comparisons between our Fast-TAD: BISTer-$1^{2\times3}$ and STAR-3×2 BISTer [1], [3]. Fault latencies are in units of t_1, the time taken to test a PLB with only one configuration or function mapped to it.

TABLE IV

RESULTS FOR RANDOM FAULT DISTRIBUTION: DIAGNOSTIC COVERAGE AND FAULT DETECTION LATENCY COMPARISONS BETWEEN OUR FAST-TAD: BISTER-$1^{2\times3}$ AND STAR-3×2 BISTER [1], [3]. FAULT LATENCIES ARE IN UNITS OF t_1, THE TIME TAKEN TO TEST A PLB WITH ONLY ONE CONFIGURATION OR FUNCTION MAPPED TO IT.

Fault	*Diagnostic Coverage (%)*		*Fault Latency (t_1)*	
Density (%)	*Fast-TAD: BISTer-$1^{2\times3}$*	*STAR-3×2 BISTer)*	*Fast-TAD: BISTer-$1^{2\times3}$*	*STAR-3×2 BISTer)*
1	100.0	98.3	252	750
2	98.2	94.3	250	763
5	96.8	82.1	245	786
7	96.8	80.4	253	796
10	96.0	74.3	247	790
15	94.5	61.1	252	791
20	92.1	59.7	243	785
25	88.5	54.5	254	795
30	86.3	46.4	247	801

TABLE V

RESULTS FOR MODERATE FAULT-CLUSTERS: DIAGNOSTIC COVERAGE AND FAULT DETECTION LATENCY COMPARISONS BETWEEN OUR FAST-TAD: BISTER-$1^{2\times3}$ AND STAR-3×2 BISTER [1], [3] FOR K=0.75 AND K=1.00. FAULT LATENCY UNIT IS THE SAME IS IN TABLE IV.

	Cluster	*Fault*	*Diagnostic Coverage (%)*		*Fault Latency (t_1)*	
k	*Density (%)*	*Density (%)*	*Fast-TAD: BISTer-$1^{2\times3}$*	*STAR-3×2 BISTer)*	*Fast-TAD: BISTer-$1^{2\times3}$*	*STAR-3×2 BISTer)*
0.75	1	4.1	92.2	61.3	237	761
	2	8.2	91.6	56.6	239	840
	3	12.3	89.2	52.0	239	854
1.00	1	4.7	92.9	55.1	240	876
	2	9.1	90.1	52.5	231	869
	3	14.2	87.6	51.1	245	826

TABLE VI

RESULTS FOR STRONG FAULT-CLUSTERS: DIAGNOSTIC COVERAGE AND FAULT DETECTION LATENCY COMPARISONS BETWEEN OUR FAST-TAD: BISTER-$1^{2\times3}$ AND STAR-3×2 BISTER [1], [3] FOR K=0.25 AND K=0.50. FAULT LATENCY UNIT IS THE SAME IS IN TABLE IV.

	Cluster	*Fault*	*Diagnostic Coverage (%)*		*Fault Latency (t_1)*	
k	*Density (%)*	*Density (%)*	*Fast-TAD: BISTer-$1^{2\times3}$*	*STAR-3×2 BISTer)*	*Fast-TAD: BISTer-$1^{2\times3}$*	*STAR-3×2 BISTer)*
0.25	1	4.6	92.1	58.8	296	869
	2	9.3	89.7	52.9	227	823
	3	19.5	83.6	49.0	230	825
0.50	1	8.8	82.5	44.5	231	899
	2	16.9	77.2	40.0	236	878
	3	26.6	76.1	38.7	223	876

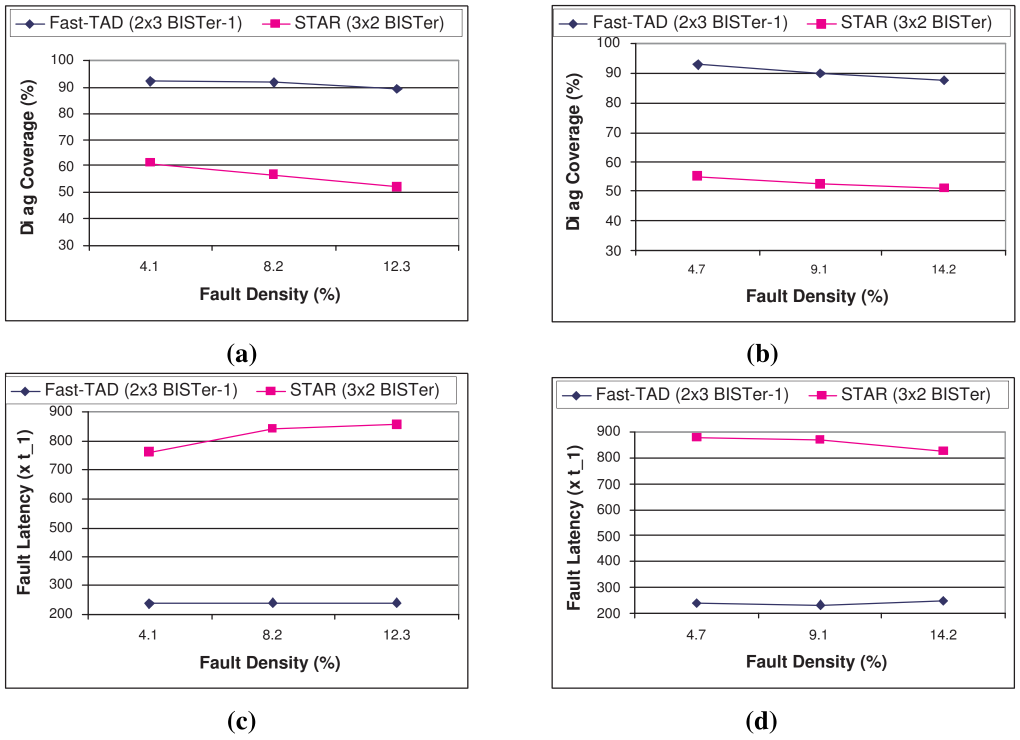


Fig. 18. Results for moderate fault-clusters: (a-b) Diagnostic coverage and (c-d) fault detection latency comparisons between our Fast-TAD: BISTer-$1^{2\times3}$ and STAR-3×2 BISTer [1], [3] for $k = 0.75$ (a,c) with and $k = 1$ (b,d). The three fault density points on the x-axis correspond to cluster densities of 1%, 2% and 3%. Fault latency unit is the same is in Fig. 17.

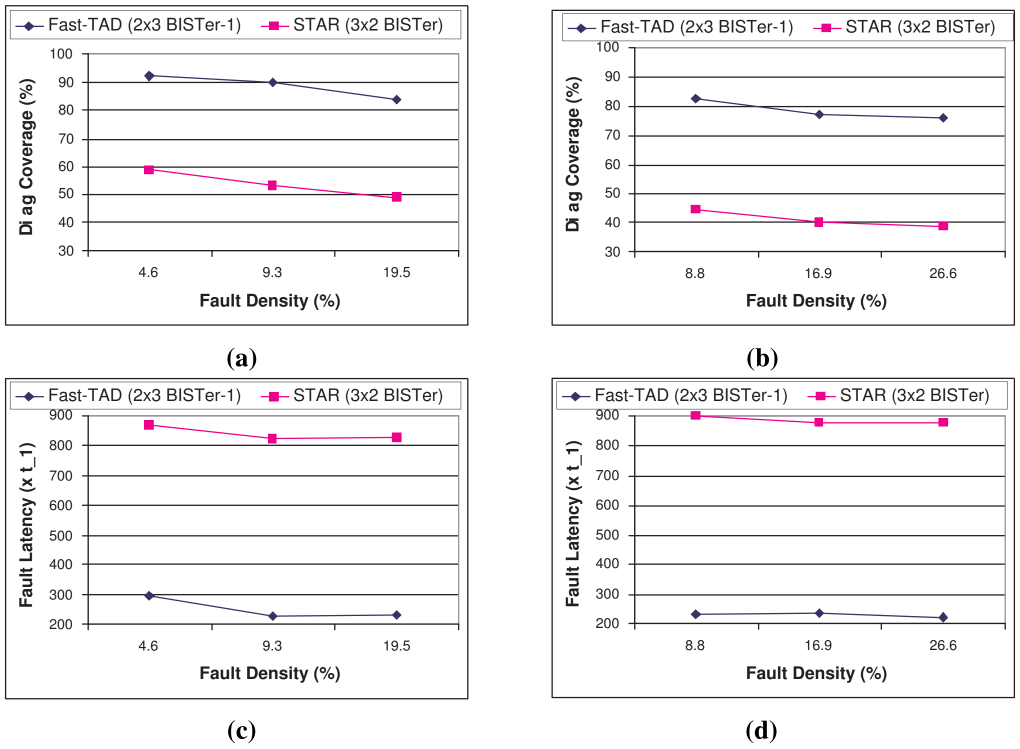


Fig. 19. Results for strong fault-clusters: (a-b) Diagnostic coverage and (c-d) fault detection latency comparisons between our Fast-TAD: BISTer-$1^{2\times3}$ and STAR-3×2 BISTer [1], [3] for $k = 0.25$ (a,c) with and $k = 0.5$ (b,d). The three fault density points on the x-axis correspond to cluster densities of 1%, 2% and 3%. Fault latency unit is the same is in Fig. 17.

FPGAs, and to future nano-technology FPGAs.

At fault densities around the 8-9% range across the different fault distributions, we also see that even BISTer-$1^{2\times3}$'s diagnosability goes down from 96% for a random fault distribution (Fig. 17(a), Table IV), to around 90% for moderate-cluster distributions (Figs. 18(a-b), Table V [$k = 0.75, 1$]), to around 83% for a strong-cluster distribution (Fig. 19(b), Table VI [$k = 0.5$]). Since a clustered-fault scenario is probably more representative of fabrication defect distributions [17], and fault densities would be high for emerging technology FPGAs, an important future research direction would entail developing BISTers that can correctly diagnose a very high percentage of faults in such distributions.

VIII. Conclusions

We presented new BISTer designs that have provable diagnosabilities of one and two, which are significant improvements over previous BIST methods. Our 2-diagnosable BISTer is the first design that offers diagnosability greater than one. We improved the testing efficiency of our BISTers for situations in which the circuit(s) mapped to the FPGA are known, by developing a fast functional test-and-diagnosis method Fast-TAD that requires a PLB to be tested for only two circuit configurations that it will provably assume under any reconfigured fault pattern, as the ROTE moves across the FPGA. This is opposed to exhaustive testing of PLBs used in previous work. Due to the provable diagnosabilities of our BISTers, it is possible to avoid time-intensive adaptive diagnosis schemes without significantly compromising diagnostic coverage, thus making our methods additionally faster. We extended our basic BISTer designs to those with multiple-PLB test-pattern generators to more efficiently test the complex PLBs of current commercial FPGAs, and proved the diagnosabilities of these designs as well. Our BIST techniques were simulated in an on-line testing wrapper, though they are also applicable in an off-line testing scenario by having multiple BISTers simultaneously configured to cover the entire FPGA and by testing a PLB for only its original circuit function.

Simulation results obtained for our extended BISTer-$1^{2\times3}$ design for random as well as clustered faults with fault densities of up to 30% show high diagnostic coverages. For example, at around a 8-10% fault density we obtain diagnostic coverages ranging from 96% for random faults to 83% for strongly clustered faults. These results represent absolute improvements of about 20-38% and relative improvements of 28-68% in diagnostic coverage over the previous best BISTer design, the STAR-3×2 BISTer of [1], [3] using non-adaptive diagnosis. Further, the fault latencies of BISTer-$1^{2\times3}$ are appreciably smaller, by factors of 3-4, than than those of STAR-3×2 BISTer. Our BIST techniques should also perform similarly well for off-line testing. Our methods are thus well-suited for high diagnostic-coverage factory and field testing of current VDSM and future nano-technology FPGAs that are expected to have high fault densities.

References

[1] M. Abramovici, C. Stroud and J. Emmert, " On-Line BIST and BIST-Based Diagnosis of FPGA Logic Blocks", *IEEE Trans. on VLSI Systems*, Vol. 12, Issue 12, Dec. 2004, pp. 1284-1294.

[2] M. Abramovici and C. Stroud, "BIST-Based Test and Diagnosis of FPGA Logic Blocks," *IEEE Trans. on VLSI Systems*, Vol. 9, Issue 1, Feb. 2001, pp. 159-172.

[3] M. Abramovici, C. Stroud, B. Skaggs and J. Emmert, "Improving on-line BIST-based diagnosis for roving STARs", *Proc. 6th IEEE Int'l On-Line Testing Workshop*, 2000, pp. 31-39.

[4] M. Abramovici, C. Stroud, S. Wijesuriya and V. Verma, "Using Roving STARs for On-Line Testing and Diagnosis of FPGAs in Fault-Tolerant Applications", *Proc. IEEE Int'l Test Conf.*, Sept'99.

[5] M. Butts, A. DeHon, S.C. Goldstein, "Molecular Electronics: Devices, Systems and Tools for Gigagate, Gigabit Chips", *International Conference on Computer Aided Design*, 2002.

[6] S. Dutt, V. Verma and H. Arslan, "A Search-Based Bump-and-Refit Approach to Incremental Routing for ECO Applications in FPGAs", *Trans. Design Autom. of Electronic Syst.*, 7(4), pp. 664-693, 2002.

[7] S. Dutt, V. Shanmugavel and S. Trimberger, "Efficient Incremental Rerouting for Fault Reconfiguration in Field Programmable Gate Arrays", *Proc. IEEE Int. Conf. Comput.-Aided Design*, 1999.

[8] S.C Goldstein and M. Budiu, "NanoFabrics: Spatial Computing Using Molecular Electronics, *Int'l Symp. Comp. Arch.*, 2001.

[9] F. Hanchek and S. Dutt, "Methodologies for Tolerating Logic and Interconnect Faults in FPGAs," *IEEE Trans. Comp.*, Special Issue on Dependable Comput., Jan. 1998, pp. 15-33.

[10] W. K. Huang, F.J. Meyer, X. Chen and F. Lombardi, "Testing Configurable LUT-Based FPGAs", *IEEE Trans. VLSI Systems*, Vol. 6, No. 2, pp. 276-283, June 1998.

[11] T. Inoue and H. Fujiwara, "Universal Fault Diagnosis for Lookup Table FPGAs," *IEEE D & T of Computers*, Vol. 15, No. 1, Jan. 1998.

[12] W. Kuo and T. Kim, "An Overview of Manufacturing Yield and Reliability Modeling for Semiconductor Products", *Proceedings of the IEEE*, Vol. 87 No. 8, Aug. 1999.

[13] J. Lach, W. H. Mangione-Smith, and M. Potkonjak, "Low Overhead Fault-Tolerant FPGA Systems," *IEEE Transactions on VLSI Systems*, Vol. 6, No. 2, 1998.

[14] N.R. Mahapatra and S. Dutt, "Efficient Network-Flow Based Techniques for Dynamic Fault Reconfiguration in FPGAs", *Proc. 29th Int'l Symp. on Fault-Tolerant Comput.*, 1999.

[15] F.P. Preparata, G. Metze and R.T. Chen, "On the connection assignment problem of diagnosable systems", *IEEE Trans. Electron. Comput.*, vol. EC-16, Dec. 1967, pp. 848-854.

[16] N.R. Shnidman, W. H. Mangione-Smith, and M. Potkonjak, "On-line Fault Detection for Bus-Based Field Programmable Gate Arrays," *IEEE Trans. on VLSI Systems*, Vol. 6, No. 4, pp. 656-666, Dec. 1998.

[17] C.H. Stapper, "The effects of wafer to wafer defect density variations on integrated circuit defect and fault distributions," *IBM Journal of Research and Development*, vol. 29, pp. 87-97, Jan. 1985.

[18] C. Stroud et al., "On-Line BIST and Diag. of FPGA Interconnect Using Roving STARs", *Proc. IEEE Int'l On-Line Test Wkshp*, 2001.

[19] V. Suthar and S. Dutt, "Efficient On-line Interconnect Testing in FPGAs with Provable Detectability for Multiple Faults", *Proc. DATE'06*, pp. 1165 - 1170, March 2006.

[20] V. Verma, S. Dutt and V. Suthar, "Efficient On-line Testing of FPGAs with Provable Diagnosabilities", *Proc. Design Automation Conf. (DAC)*, pp. 498-503, 2004 (**nominated for a best paper award**).