Efficient On-Line Testing of FPGAs with Provable Diagnosabilities *

Vinay Verma[1], Shantanu Dutt[2] and Vishal Suthar[2]
[1]Xilinx Inc., San Jose, CA and [2]Dept. of ECE, University of Illinois at Chicago

Abstract
We present novel and efficient methods for on-line testing in FPGAs. The testing approach uses a ROving TEster (ROTE), which has provable diagnosabilities and is also faster than prior FPGA testing methods. We present 1- and 2-diagnosable built-in self-tester (BISTer) designs that make up the ROTE, and that avoid expensive adaptive diagnosis. To the best of our knowledge, this is the first time that a BISTer design with diagnosability greater than one has been developed for FPGAs. We also develop functional testing methods that test PLBs in only two circuit functions that will be mapped to them (as opposed to testing PLBs in all their operational modes) as the ROTE moves across a functioning FPGA. Simulation results show that our 1-diagnosable BISTer and our functional testing technique leads to significantly more accurate (98% (90.5%) fault coverage at a fault/defect density of 10% (25%)) and faster test-and-diagnosis of FPGAs than achieved by previous work. In general, it is expected that ROTE will achieve high fault coverages at fault/defect densities of up to 25% using our 1-diagnosable BISTer and up to 33% using our 2-diagnosable BISTer. Our methods should thus prove useful for testing current very deep submicron FPGAs as well as future nano-CMOS and molecular nanotechnology FPGAs in which defect densities are expected to be in the 10% range.

Categories and Subject Descriptors:
B.7 Integrated Circuits, B.7.0 General, B.8 Performance and Reliability: B.8.1 Reliability, Testing, and Fault-Tolerance

General Terms:
Algorithms, Design, Reliability

Keywords and Phrases:
built-in self-tester (BISTer), diagnosability, FPGAs, functional testing, on-line testing, roving tester (ROTE).

1 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 as well as decreased reliability in 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—they increase device fabrication yield (thus lowering costs) and the reliability of operation of FPGAs in platforms ranging from mission/life-critical systems to commercial products.

Testing and diagnosis is integral to providing increased reliability in FPGA-based systems. Off-line testing methods for FPGAs are reasonably mature and well developed [8, 9, 15]. Off-line testing is acceptable in application environments where there are little real-time or time-to-market constraints on the device or system being 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. The techniques we present here are primarily meant for on-line testing.

We also 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 FF(s), etc. This will be required when it is not known which circuit(s) will be mapped to the FPGA, as for 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, for 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.

There have only been a few methods proposed for on-line FPGA testing, including [1, 2, 13, 14]. In [13], 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 [2] uses a roving tester called STAR to test a portion of the chip exhaustively (irrespec-

*This work was funded partly by a grant from Xilinx Corp. and partly by Darpa Contract # F33615-98-C-1318.

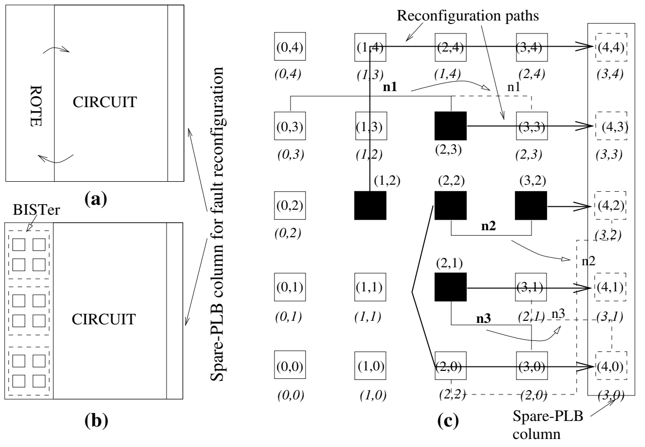


Figure 1: *(a) Roving concept. (b) BISTer tiles in ROTE area. (c) Four reconfiguration paths (shown by dark arrows) for four faulty PLBs and net re-routings (shown by dashed lines) of nets connected to PLBs on these paths. The new PLB labels in the reconfigured FPGA are shown in italics below each PLB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

tive 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. 3, the basic built-in self tester (BISTer) in [2] 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 always performs exhaustive testing of PLBs (even when the circuit is known), it can be very time consuming. In [1], a 3×2 BISTer was proposed in combination with the STAR tester of [2] 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 identified); it also diagnoses some but not all patterns of two PLB faults. A BISTer for interconnects is presented in [14].

We present here a novel 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 diagnosability, thus avoiding expensive adaptive diagnosis. Further, in the "known-circuit" scenario, a fast functional test-and-diagnosis method is proposed that tests a PLB only in its possible circuit functions. Our testing methods are thus more accurate (higher error coverage and fault diagnosability) and faster than those developed in previous work. In this paper we address PLB faults; interconnect fault test-and-diagnosis will be addressed in future work.

The rest of the paper is organized as follows. Section 2 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. 3 we establish that a previous BISTer design is 0-diagnosable. Section 4 discusses two new BISTer architectures with provable diagnosability, while Sec. 5 presents 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. Section 6 present our simulation results, and conclusions are in Sec. 7.

2 The Big Picture – Roving Tester and Fault Reconfiguration

We present here the overview of the process in which the FPGA is repeatedly tested 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 field testing. For factory testing, the process would conclude with a single roving of the tester across the FPGA after which each PLB would have been tested and diagnosed once.

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, 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 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. 4.1) in it occupies a 2×2 array of PLBs whose latest diagnosis status is "fault-free". This is shown in Fig. 5, which also illustrates Lemma 1 in Sec. 5.

When fault(s) are detected and diagnosed, then using techniques proposed in [5, 11], each faulty PLB is reconfigured directly or indirectly using a unique spare PLB. Briefly, the steps are:

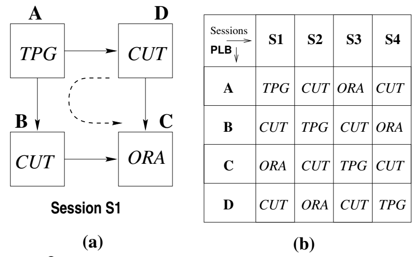
1. Compute *reconfiguration paths* from each identified faulty PLB to a spare PLB using fast network flow algorithms presented in [11]. PLB v following PLB u in a reconfiguration path means that v will be configured with u's functionality for each such (u, v) pair to achieve reconfiguration; see Fig. 1(c).
2. Perform incremental re-routing (e.g., [4, 5]) to extend/reroute 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.

3 Previous BISTer Design

In this section, we present the BISTer design of [2] denoted here by BISTer-0, and prove that it is 0-diagnosable[1]. This 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

[1]While BISTer-0 has been superseded by the 1-diagnosable 3×2 BISTer of [1] by the same primary designers of BISTer-0, BISTer-0 provides a starting point for us for developing our own 1-diagnosable BISTer that has better fault coverage than the the 3×2 BISTer of [1] as we show in Sec. 6.

faulty interconnects has been addressed in [14], and will also be addressed by us in future work to apply to our framework.



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*

Figure 2: *(a) BISTer-0 architecture of [2]. (b) Cycling of PLBs in BISTer-0 tile.*

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 [2] but, as we show later, it cannot diagnose, i.e., locate, any of them—it is 0-diagnosable . Thus expensive adaptive diagnosis schemes are used in [2] even for a single PLB fault.

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.

A *gross syndrome* of a session is the overall *pass/fail* (indicated as"X"/"√" in Tables 1- 3) observation over all modes of operation 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 "√" (*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. ⋄

4 New BISTer Architectures

We now present our new BISTer designs that have non-zero diagnosis capabilities—BISTer-1 which is 1-diagnosable and

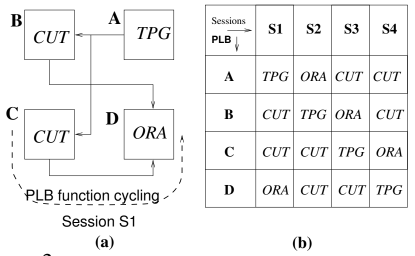
Ses → S.No ↓	S_1	S_2	S_3	S_4	Inference
1	√	√	√	√	No faulty PLB
2	X	√	√	√	Fault not in PLB
3	√	X	√	√	Fault not in PLB
4	√	√	X	√	Fault not in PLB
5	√	√	√	X	Fault not in PLB
6	X	X	√	√	Faulty C *(CUT)*
7	√	X	X	√	Faulty D *(CUT)*
8	√	√	X	X	Faulty A *(CUT)*
9	X	√	√	X	Faulty B *(CUT)*
10	X	√	X	√	Fault not in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

Table 1: *Gross syndromes for BISTer-1 and their diagnosis under assumption of at most one faulty PLB.*

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 used in [2] 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% defect/fault density[2], and 2 faults out of 6 PLBs in BISTer-2 or a 33% defect/fault density. In this section, we present the 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.

4.1 A one-diagnosable BISTer

Figure 3 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.



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*

Figure 3: *(a) Our BISTer-1 architecture. (b) The four sessions in BISTer-1.*

As shown in Fig. 3, 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.

Table 1 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 1) 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

[2]A defect/fault density of x% means that x% of all the PLBs are defective/faulty.

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 *pass* ("$\surd$"), 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.

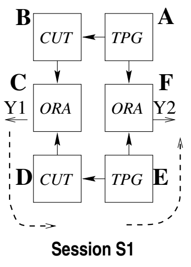
Case 6: (Row 16). In this case, all the sessions report *fail*. This case is not possible with at most one faulty PLB, since when the faulty PLB is configured as a TPG, the gross syndrome should be a *pass*.

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: 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 [12] 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. ◇



(a)

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*

(b)

Figure 4: *(a) BISTer-2 architecture. (b) Six sessions of BISTer-2.*

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

Table 2: *Gross Y_1, Y_2 syndromes for BISTer-2 for one faulty PLB assuming that a faulty PLB also fails as a TPG and as an ORA.*

4.2 A two-diagnosable BISTer

The BISTer-2 architecture which has six PLBs is shown in Fig 4a. 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. 4b 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 as a TPG. Thus according to the PMC model [12] 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 for the case that the faulty PLB also fails as a TPG and ORA; the proof for the case it passes as a TPG and as an ORA is similar. Table 2 which is self-explanatory shows that the gross syndromes Y_1 for each faulty PLB is unique. 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*. With this scenario, the assumption that fault masking does not occur for all detailed syndromes is a very high probability one. Table 3 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. 4b. 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: $dS_2 = dS_6$, $dS_3 = dS_5$.

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

Table 3: *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).*

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 2 (1 faulty PLB) are distinct from those of Table 3 (2 faulty PLBs). We can thus distinguish between the syndromes for single and double faults. ◇

5 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 section detect 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. 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.

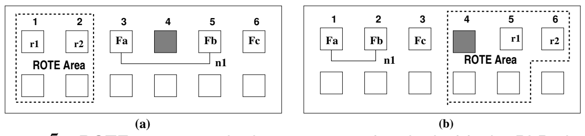


Figure 5: *ROTE movement in the presence of a single f-faulty PLB shown dark. F_i represents the functionality of each PLB. (a) Initial position of the ROTE. PLB 3 implements its* original function *F_a; PLB 6 is 2 fault-free PLBs to its right. (b) ROTE occupies columns 4 and 5. Note the warped shape 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.*

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 other words, x_2 is the original function of the PLB two fault-free PLBs to X's right in the same row)[3]; see Figs. 5. x_1 and x_2 are called X's *operational functions/configurations*. 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.

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)

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

(c)

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

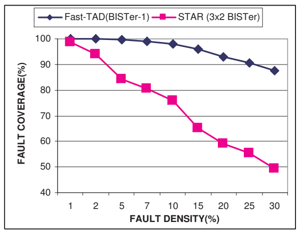
(d)

Figure 6: *(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.*

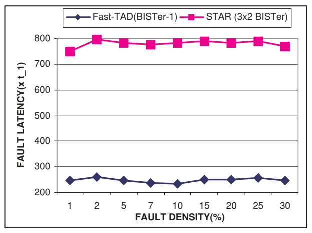
Fast-TAD uses the BISTer-1 architecture of Sec. 4, but tests each PLB only in its operational configurations. We use two sets of tests. The first test set, uses all four BISTer-1 sessions, while in the second test set, which is used for further diagnosis, only one session is adaptively used. In the first 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 6a shows all the sessions in the first test set, along with the functions configured in the CUTs. Figure 6c shows the gross syndrome for PLB configurations in Fig. 6a. 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. 6b) 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. 6c). Only one further session of BISTer-1 is needed to distinguish between the above PLBs. As shown in Fig. 6d, 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.

[3]Lemma 1 establishes the two functions x_1 and x_2 that PLB X would implement in any ROTE position and under any reconfigured fault pattern.



(a)



(b)

Figure 7: *(a) Fault coverage and (b) fault detection latency comparisons between our Fast-TAD(BISTer-1) and STAR(3×2 BISTer) [1] testers for random fault distributions. Fault latencies are in units of t_1, where t_1 is the time taken to test a PLB with only one configuration or function mapped to it.*

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. 5 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. $\diamond$

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. 6c, 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. 6c, we may not be able to distinguish between f-faulty PLBs A, C and between f-faulty PLBs B, D. The second test (Fig. 6b) 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. 6b 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. 6b is performed). We see from Fig. 6d 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. $\diamond$

6 Simulation Results

A 32×32 FPGA array with 2-input 1-output PLBs was functionally simulated using C, with carry-lookahead adders mapped to it. Test and diagnosis using two techniques was implemented on this FPGA: (1) Fast-TAD with BISTer-1 with the small modification that in each ROTE position, except for the bottom BISTer-1, each BISTer-1 is shifted down by one row for another testing phase after its testing phase (4 sessions) is completed in its original position; this is done so that every 2×2 PLB subarray in the FPGA is tested by a BISTer-1 tile. (2) The STAR rover with the 1-diagnosable 3×2 BISTer of [1] that performs exhaustive testing.

Figure 7 shows the fault coverages and fault detection latencies of the two BISTer methods for randomly distributed faults at different fault densities. As can be seen in the plots, our Fast-TAD method using BISTer-1 far outperforms the STAR method using a 3×2 BISTer in both metrics across different fault densities ranging from 1% to as high as 30%. Our technique is quite stable across fault densities, giving coverages of 98% and 90.5% at 10% and 25% fault densities, respectively, while the STAR method's coverage falls rapidly to about 76% at 10% fault density and to 55.5% at a fault density of 25%. Also, fault detection latency of the STAR method is about 3 times more than that of our technique.

7 Conclusions

We presented new BISTer designs that have provable diagnosabilities of one and two, which are significant improvements over previous on-line BIST methods. To the best of our knowledge, our 2-diagnosable BISTer is the first design that offers diagnosability greater than one. Further, for efficient and accurate testing in scenarios where the circuit(s) mapped or to-be mapped to the FPGA are known, a fast functional test-and-diagnosis method Fast-TAD was developed that does not require a PLB to be exhaustively tested as in previous work; in this method a PLB is tested in only two circuit configurations it will assume under any reconfigured fault pattern, as the ROTE moves across the FPGA. All our test-and-diagnosis techniques also avoid expensive adaptive diagnosis, assuming at most one fault in a BISTer-1 tile with four PLBs or two faults in a BISTer-2 tile with six PLBs. This means that we can detect and locate randomly distributed defects or faults with high probability in the presence of defect/fault densities of up to 25% (when using Bister-1) and up to 33% (when using Bister-2). Simulation results for fault densities of up to 30% support our above expectation for BISTer-1 and established our much higher error coverage and faster fault detection compared to the previous best technique of [1]. Our techniques are thus well-suited for reliable post-fabrication testing and field operation of current VDSM as well as future nano-technology FPGAs that are expected to have defect densities of about 10%.

References

[1] 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.

[2] 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.

[3] M. Butts, A. DeHon, S.C. Goldstein, "Molecular Electronics: Devices, Systems and Tools for Gigagate, Gigabit Chips", *ICCAD'02*.

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

[5] 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.

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

[7] 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.

[8] 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.

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

[10] 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.

[11] 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.

[12] 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.

[13] 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*, pp. 656-666, Dec. 1998.

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

[15] C. Stroud, E. Lee and M. Abramovici, "BIST-Based Diag. of FPGA Logic Blocks," *Proc. IEEE Int'l Test Conf.*, pp. 539-547, 1997.