

BIST-Based Test and Diagnosis of FPGA Logic Blocks¹



Miron Abramovici

Bell Labs - Lucent Technologies
Murray Hill, NJ

Charles Stroud²

Dept. of Electrical and Computer Engineering
University of North Carolina at Charlotte

Keywords: Built-In Self-Test, FPGA testing, FPGA diagnosis, fault-tolerance, reconfigurable systems

Abstract: We present a Built-In Self-Test (BIST) approach able to detect and accurately diagnose all single and practically all multiple faulty programmable logic blocks (PLBs) in Field Programmable Gate Arrays (FPGAs) with maximum diagnostic resolution. Unlike conventional BIST, FPGA BIST does not involve any area overhead or performance degradation. We also identify and solve the problem of testing configuration multiplexers, that was either ignored or incorrectly solved in most previous work. We introduce the first diagnosis method for multiple faulty PLBs; for any faulty PLB, we also identify its internal faulty modules or modes of operation. Our accurate diagnosis provides the basis for both failure analysis used for yield improvement and for any repair strategy used for fault-tolerance in reconfigurable systems. We present experimental results showing detection and identification of faulty PLBs in actual defective FPGAs. Our BIST architecture is easily scalable.

1. Introduction

An FPGA consists of an $N \times N$ array of programmable logic blocks (PLBs) and programmable I/O blocks, connected by a programmable interconnect network. In this paper, we consider RAM-based FPGAs, which are programmed by writing an on-chip configuration memory. FPGA manufacturing tests should detect all the faults affecting every possible mode of operation of its PLBs, and also detect all the faults affecting its interconnect network. Usually the logic and the interconnect are separately tested. The goal of “detecting a faulty PLB” implicitly assumes that a defective PLB may have multiple internal faults. The usual assumption in logic test is that the FPGA under test has at most one faulty PLB. However, multiple faulty PLBs may exist in newly manufactured FPGAs or may appear in FPGAs used in long missions in harsh environments. Interconnect testing targets faults such as shorts, opens, and programmable switches stuck-on and stuck-off.

Most previous methods for FPGA testing, both for PLBs [22][17][21][18][19][32][33][20] and for interconnect [25][30][31][9], rely on externally applied vectors, hence they are applicable only for device-level testing. Therefore, while these tests can be used for manufacturing test, they are not applicable to in-system test or to fault-tolerant system applications. In contrast, BIST-based methods [37][38][39][7][40]

1. This material is based upon work supported in part by the National Science Foundation under Grant No. MIP-9409682, by the DARPA ACS program under contract F33615-98-C-1318, by the Univ. of Kentucky Center for Robotics and Manufacturing Systems, and by the Microelectronics Group of Lucent Technologies.

2. Formerly with the Dept. of Electrical Engineering, University of Kentucky.

[41][42][28][13][15][4] can also be reused for board and system-level testing; their reuse reduces the effort involved in developing system diagnostic routines to test FPGAs in their system mode of operation. Since BIST deals with every FPGA in isolation, it also provides a simple solution to the problem of locating faulty FPGAs in the system. Of course, BIST is more difficult to implement, because, unlike in external testing, we cannot rely on a fault-free tester to provide vectors and analyze results. Following the approach introduced in [37][38], BIST methods configure one part of the FPGA to be under test, and the other part to generate vectors for, and to analyze the results from, the subcircuits under test; then the resources of the FPGA change roles so that the entire FPGA is eventually tested. BIST techniques have also been applied to on-line FPGA testing [34][2][3], but in this paper we discuss only off-line testing. Other on-line FPGA test methods rely on redundant design techniques [6][10].

Both external-test and BIST methods involve multiple configurations of the FPGA. An FPGA that is tested in-system is configured via its boundary-scan interface [35]. The data required to reconfigure the FPGA under test are maintained within the test environment - automatic test equipment(ATE) for device test, or CPU (or maintenance processor) for in-system test. To minimize the memory requirements as well as the test time (which is dominated by the device programming time), the number of test configurations should be kept to a minimum. BIST methods may require more configurations than ATE-based FPGA tests.

Conventional BIST approaches introduce both area overhead and delay penalties; the latter may result in speed degradation unacceptable in high-performance systems. In contrast, BIST for FPGAs - first introduced for testing PLBs [37][38] and then extended to testing programmable interconnect [41] - exploits the reprogrammability of an FPGA to configure it exclusively with BIST logic during off-line testing. In this way, for manufacturing test, *testability is achieved without any cost*, since the BIST logic “disappears” when the FPGA is no longer under test. When BIST is complete, an FPGA tested in-system needs to be reconfigured for its normal operation. For in-system test, the only cost is the additional memory required for the BIST configurations. The results of our implementation will show that this cost is negligible.

Diagnosis consists of mapping an incorrect response from the circuit under test into the fault(s) that can explain the obtained response. The required diagnostic resolution depends on the goal of the testing process. Usually in system-level testing, the objective is to locate a replaceable defective component. Thus, in-system identification of a faulty FPGA would be sufficient in this context. However, one can take advantage of the reprogrammability and the regular structure of an FPGA to achieve fault tolerance by repairing the FPGA in place. Here the goal is to assure that the defective chip will still correctly execute its intended function. This is much more economical than replacing defective FPGAs, and it is an essential feature in environments where device replacement is not feasible or practical, such as unmanned missions or remote stations. It is interesting to note that in the Teramac custom computer [7], about 75% of the 864 FPGAs used in the system are defective and have been repaired by fault-tolerant reconfiguration. This process requires the accurate identification of faulty PLBs, which are bypassed and replaced with fault-free unused cells by reprogramming the FPGA (if such cells are still available) [23][29][7][11]. The same resolution (to the level of a faulty PLB) is required for the “node-covering” fault-tolerant technique, where the repair occurs after manufacturing testing and is application-independent and invisible to the user [14]. Another yield-enhancement technique [16] replaces an entire faulty row (or column) by a spare one, and

hence its resolution requirement is only to identify a faulty row (or column). When the goal of testing is the improvement of the manufacturing process, then the most accurate resolution - locating faults inside a PLB - is required to support subsequent failure analysis. The ability to locate defective modules inside a PLB enables a new form of fault-tolerance that reuses the fault-free modules or fault-free modes of operation of partially defective PLBs [2][11]. Other fault-tolerance techniques [24] also rely on identifying faults within defective PLBs (referred to as “clusters”) with the goal of reusing them.

Many FPGA test methods configure the FPGA to create *iterative logic arrays* (ILAs), composed of identical cells connected serially as one-dimensional horizontal or vertical arrays [23][17][39][40][18][21][19][28][32][33][20] or 2-D arrays [22]. Sometimes separate ILAs are used to propagate errors from the logic under test [39][40][20]. Constructing ILAs to test the RAM operation of PLBs is described in [39][18][32][33]. ILAs are especially useful when they are *C-testable*, since then they can be completely tested with a number of tests that does not depend on the number of cells in the ILA. An external-test ILA-based method has been used to detect multiple faulty PLBs [20]. ILA structures are also useful for diagnosing single faulty PLBs; typically, testing the horizontal (vertical) ILAs identifies a faulty row (column), and the faulty PLB is located at the intersection of the faulty row and the faulty column [23][40][21][28]. However, such procedures may not be reliable in the presence of multiple faults, since a fault-identifying signal may subsequently propagate through a defective block.

The method used to check the (non-commercial) FPGAs used in the Teramac custom computer [7] configures each row as a pseudo-random sequence generator and checks the final register contents after applying a given number of clock cycles against an expected signature. The same procedure is then repeated using columns instead of rows, and the faulty cells are located at the intersection of the faulty rows with the faulty columns. However, the test provided to the blocks that form the pseudo-random sequence generator does not achieve complete fault coverage and hence it cannot guarantee accurate diagnosis (in general, fault detection is a necessary condition for fault location). In addition, applying the test requires a fault-free finite-state machine in the FPGA, the total test time is very large, and developing the diagnostics tests is an expensive manual process.

The BIST method of [42] provides hierarchical and adaptive diagnosis; it first identifies multiple faulty groups of PLBs, and then the faulty PLBs in the faulty groups. For 2^{t+1} groups of PLBs, this approach can diagnose only up to t faulty groups. To identify faulty PLBs, the PLBs in faulty groups are compared with PLBs from fault-free groups. This approach requires a large number of test configurations, which makes the total test time prohibitive for manufacturing testing. Scalability appears to be a problem, as the layout is not regular, and the number of configurations increases with the size of the FPGA.

In this paper, we present *the first complete test and diagnosis method for FPGAs*. Our BIST technique detects any single faulty PLB and any combination of multiple faulty PLBs, without requiring any fault-free core in the FPGA; the tests are complete for almost any fault model. We have identified and solved the problem of testing configuration multiplexers, that was either ignored or incorrectly solved in most previous work. Our new diagnosis algorithm locates any single faulty PLB and, except for few pathological cases, identifies any possible combination of multiple faulty PLBs. Moreover, it also determines the faulty modules or faulty modes of operation of any defective PLB. Our approach is easily scalable. The same BIST approach can perform fault detection and identification at any level of testing - device level for man-

ufacturing testing and yield enhancement, and system level for repair strategies in fault-tolerant applications. We used the Lucent Optimized Reconfigurable Cell Array (ORCA) [26] for the initial design and implementation of the BIST-based diagnostic approach, but we emphasize that our technique can be applied to other RAM-based FPGAs, including Xilinx [43] and Altera [5].

Since our BIST is independent of the system function implemented in the FPGA, our approach could be considered an “overkill” for in-system testing: why don’t we test the FPGA just in its normal mode of operation? The main reason is that during the normal operation of an adaptive system, of a custom-computing machine, or of a fault-tolerant application, the same FPGA will be used with different configurations at different times. Hence, our strategy makes certain that no assigned function will be incorrectly performed because of *dormant faults* that affect currently unused logic cells or unused modes of operation of the active cells. This is important because testing of dormant faults is essential in achieving high reliability in safety-critical applications [36].

The remainder of this paper is organized as follows. Section 2 outlines our BIST architecture. Section 3 reviews the method used to access the BIST architecture via the FPGA Boundary Scan interface. Section 4 presents the fault detection capabilities of our BIST approach, while Section 5 analyzes its diagnostic features. Section 6 discusses the results from the successful use of this approach in locating faulty PLBs in manufactured FPGAs. Finally, Section 7 presents our conclusions.

2. The BIST Architecture

2.1 Testing PLBs

The strategy of our FPGA BIST approach is to configure groups of PLBs as *test pattern generators* (TPGs) and *output response analyzers* (ORAs), and another group as *blocks under test* (BUTs), as illustrated in Figure 1a. The BUTs are then repeatedly reconfigured to test them in all their modes of operation. We refer to the test process that occurs for one configuration as a *test phase*. A *test session* is a sequence of test phases that completely test the BUTs in all of their modes of operation. Once the BUTs have been tested, the roles of the PLBs are reversed so that in the next test session the previous BUTs become TPGs or ORAs, and vice versa. Since half of the PLBs are BUTs during each test session, we need only two test sessions to test all PLBs in the FPGA. Figures 1b and 1c show the floorplans for the two test sessions - called *NS* and *SN* - that completely test every PLB in an 8x8 FPGA. Figure 1a corresponds to the first four

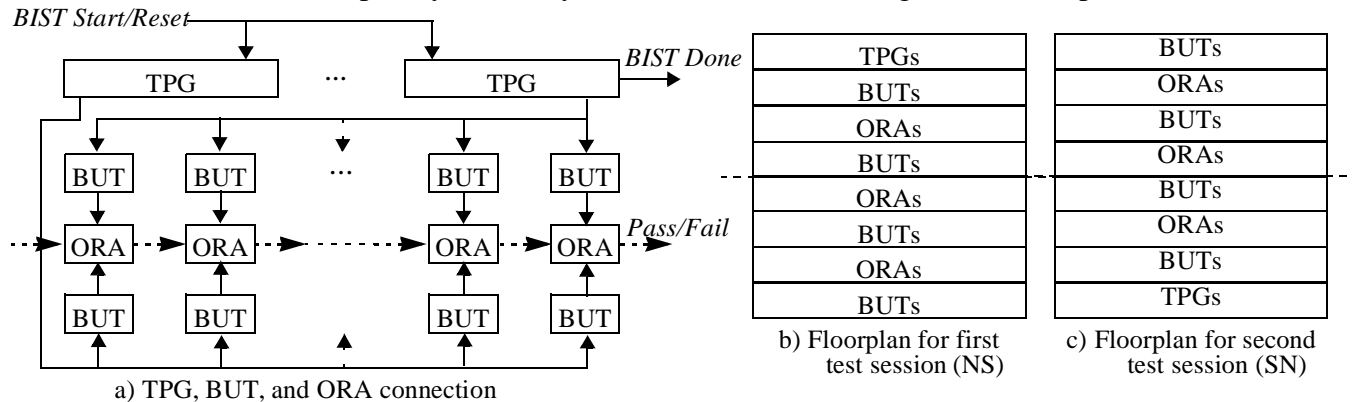


Figure 1. The BIST architecture.

rows in Figure 1b. The name of a session denotes the direction of the flow of test patterns during that session. The floorplan for *SN* is obtained by flipping the floorplan for *NS* around the horizontal axis shown as a dotted line in the middle of the array. This changes the roles of the PLBs so that every PLB is under test in one of the two test sessions. Note that all BUTs are tested in parallel and that patterns may be applied at-speed.

Each test phase consists of the following steps: 1) reconfigure the FPGA, 2) initiate the test sequence, 3) generate test patterns, 4) analyze output responses, and 5) read the test results. In step 1, the test controller (ATE for wafer/package testing; CPU or maintenance processor for board/system testing) interacts with the FPGA(s) under test to reconfigure the logic by retrieving a BIST configuration from the configuration storage (ATE memory; disk) and loading it into the FPGA(s). The test controller also initializes the TPGs, BUTs, and ORAs and initiates the BIST sequence (via the *BIST Start/Reset* input in step 2) and reads the subsequent *Pass/Fail* results (step 5). Steps 3 and 4 are concurrently performed by the BIST logic within the device. After the board or system-level BIST is complete, the test controller must reconfigure the FPGA for its normal system function; hence the normal device configuration must be stored along with the BIST configurations. The test application time is dominated by the FPGA reconfiguration time. Since the total test and diagnosis time is a major factor in the system down time (system availability) and cost, an important goal of our BIST approach is to minimize the number of configurations used for test and diagnosis.

Figure 2 illustrates the typical structure of a PLB, consisting of a memory block that can function as a look-up table (LUT) or RAM, several flip-flops (FFs), and multiplexing output logic. The LUT/RAM block may also contain special-purpose logic for arithmetic functions (counters, adders, multipliers, etc.) The RAM may be configured in various modes of operation (synchronous, asynchronous, single-port, dual-port, etc.). The FFs can also be configured as latches, and may have programmable clock-enable, pre-set/clear, and data selector functions. Our strategy relies on *pseudoexhaustive testing* [27], which in this context means that every subcircuit of a PLB is tested with exhaustive patterns in each one of its modes of operation [1]. The memory block is checked with RAM test sequences which are exhaustive for faults specific to RAMs [12]. Note that all three subcircuits of a PLB are easily controllable and observable from the PLB's I/O pins, and that exhaustive testing of every module is feasible since the number of inputs is reasonably small. This results in practically complete fault coverage without explicit fault model assumptions and without fault simulation. For example, all single and multiple stuck-at faults, as well as all faults (of any type) that do not increase the number of states, are guaranteed to be detected; the overwhelming majority of faults that do increase the number of states are also detected. Thus *for any practical purpose the PLB logic test is complete*. (Applying a complete logic test at normal operating frequency is likely to create a good delay-fault test, but in this paper we deal only with logic faults.)

In every test phase, a BUT is configured in a different mode of operation; hence its pseudoexhaustive

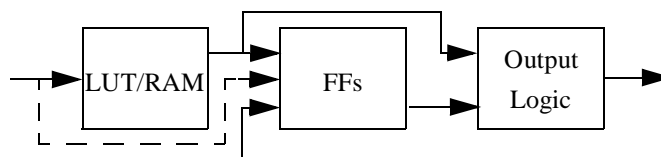


Figure 2. Typical PLB structure

test may also change from phase to phase. For example, the test sequence for combinational logic followed by FFs is different from the test sequence for a RAM. Thus a TPG may have different structures depending on the sequences needed in different phases. All BUTs are configured to have the same logical function and receive the same input test patterns from the two identical TPG blocks. Since all fault-free BUTs must produce the same output patterns, the ORAs simply compare corresponding outputs from different BUTs. Unlike the signature-based compression circuits found in most BIST applications, comparator-based ORAs do not suffer from the aliasing problem that occurs when a faulty circuit produces the good circuit signature.

Figure 3 shows the structure of an ORA comparing four pairs of BUT outputs. The signal $BuOi$ ($BdOi$) is the i -th output from the BUT up above (down below) the ORA. The FF stores the result of the comparison, and the feedback loop latches the first mismatch in the FF. This represents a compression of the results, since any number of mismatches in the same phase translate into one error. The result FFs of all ORAs are connected to form a scan chain (indicated by the dotted lines in Figure 1a). In every test phase, the scan chain is also tested, to assure the integrity of the test results. Our previous BIST approach [40] recorded only one *Pass/Fail* result for every row of ORAs. However, storing the test results of each ORA cell significantly improves the diagnosis resolution achievable by this architecture, as we will show in Section 5. For an $N \times N$ FPGA, the number of ORA cells is $N_{ORA} = (N^2/2) - N$.

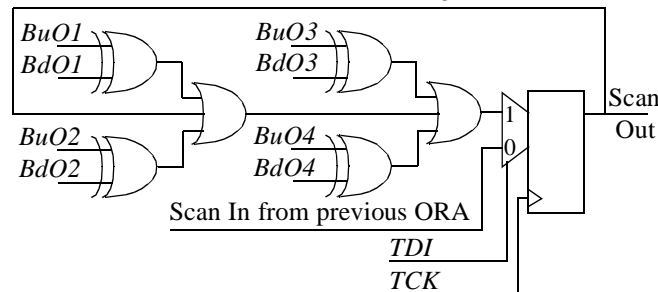


Figure 3. Integrated ORA/scan cell

Two important features of this architecture help both testing and diagnosis of the FPGA. First, every BUT (except those in the first two and in the last two rows) is simultaneously compared with two other BUTs by two different ORAs (one above and one below). Second, the pair of BUTs being compared by each ORA are fed by two different TPGs. Section 4 and Section 5 will show how these features help achieve complete testing and maximum diagnostic resolution.

2.2 Testing Configuration Multiplexers

A configuration multiplexer (MUX) is a commonly used hardware mechanism that selects subcircuits for various modes of operation. A configuration MUX is controlled by configuration memory bits to select one input to be connected to its output. In Figure 4a, assume that we set the configuration bit S to 0 to connect $V0$ to X . Then the subcircuit producing $V1$ disappears from the circuit model seen by the user. This is correct from a design viewpoint, because the value $V1$ can no longer affect X in the current configuration. But from a testing viewpoint, in any test for the MUX, we need to set $V0$ and $V1$ to complementary values. In general, for a MUX with k inputs, if V is the value of the selected input, all the other $k-1$ inputs should be set to value \bar{V} .

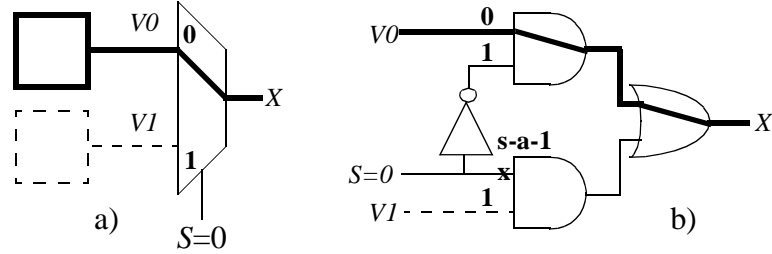


Figure 4. C onfiguration multiplexer

The problem arises because FPGA CAD tools generate the configuration bitstream based on the user model, which will never include the functionally inactive subcircuits (called “invisible logic” in [38]). Thus in Figure 4a, when $S=0$, $V0$ will be set to both 0 and 1, but VI cannot change. Similarly, the user logic cannot control $V0$ in any configuration where $S=1$. The result is that the testing of the MUX may not be complete. For example, the $s-a-1$ fault in the gate-level MUX model in Figure 4b is detected only when $S=0$, $V0=0$, and $VI=1$. But this pattern may never be applied if VI cannot be controlled when $S=0$.

Our solution relies on separately configuring the invisible logic so that it will generate the proper values needed for the inactive MUX inputs. Then we “overlay” the resulting configuration files over the main configuration file with the active logic, and we “merge” them without changing any MUX setting done in the main configuration. This process is conceptually simple, but its implementation requires knowledge of the FPGA configuration stream structure.

Note that in most previous work dealing with testing FPGAs, the problem of testing a configuration MUX is either not addressed or it is “solved” functionally, by connecting every input in turn to the output, and providing both 0 and 1 values to the selected input. However, the invisible logic driving the inactive inputs is completely ignored. Hence prior claims of “complete testing” may not be valid since the testing of every configuration MUX in the FPGA is likely to be incomplete. Methods that did provide complete tests for configuration multiplexers, such as [21][19], used models that did not remove the invisible logic. But such models cannot be used with the existing FPGA CAD tools to generate configuration files.

2.3 Testing Memory Blocks

For the LUT/RAM block of a PLB, we first test its RAM mode of operation. We configure the TPGs to apply a march test [12], which detects, among other faults, all the stuck faults in memory cells, as well as all faults in the address and read/write circuitry of the RAM. We rely on this RAM-mode test as the major test of the memory block, so that subsequent tests for different modes of operation of the same block do not need to retarget the already detected faults. When we test the LUT operation, we configure alternating XOR/XNOR functions for the LUT outputs during one test phase, and alternating XNOR/XOR functions during a second test phase. For any combinational function of n inputs, the TPG will apply all 2^n vectors.

This strategy using a RAM test sequence to detect most faults in the LUT/RAM block is not applicable to FPGAs whose PLB memory block cannot be operated as RAM, and as a result, functions only as a ROM. To test a ROM-only type of LUT with n address bits, we can use the $2n$ configurations proposed in [18].

Some currently available FPGAs contain large embedded RAM arrays (much larger than the RAMs used within PLBs). To test such RAMs we need an additional session, in which the PLBs surrounding a

RAM implement the same BIST logic that would be added to test the RAM had it been embedded on a system chip. The difference is that in an FPGA the BIST logic would disappear after the RAM test session. All embedded RAMs may be tested in parallel, possibly sharing the BIST controller. If the FPGA has several identical RAM modules, we can feed them with the same patterns, and use comparators to check mismatches between corresponding outputs.

2.4 Scalability of the BIST Approach

Our BIST architecture has a very regular easily-scalable structure, automatically generated by a simple procedure (that also does algorithmic placement and routing), based on the dimensions ($N \times N$) of the FPGA array. Since an ORA compares the outputs of its two neighbor BUTs, all signals from BUTs to ORAs use only local routing resources which are oblivious to the size of the FPGA. Global routing is used to distribute the patterns generated by TPGs to BUTs. Ignoring fanout load limitations, adding rows and columns to an array of PLBs will just extend the length of the vertical and horizontal global lines fed by TPG outputs, hence the usage of the global routing resources required for distributing the TPG patterns does not change with the FPGA size. In the ORCA FPGA, we use the bidirectional drivers available in the local routing surrounding a PLB to redistribute incoming TPG signals, thus avoiding fanout overload. The following analysis is for FPGAs where such drivers are not available. If k is the number of cells used by one TPG ($k=4$ in our implementation), a TPG row has $N_{TPG} = \lfloor N/k \rfloor$ TPGs. (For simplicity we assume that N is always a multiple of k .) We divide the BUTs into N_{TPG} subsets of k columns and we feed each such subset from its closest two TPGs. Then each TPG output drives $N^2 / (2 \cdot N_{TPG}) = k \cdot N / 2$ BUTs. This shows that the loading grows only linearly with N , which is the square root of the size of the FPGA. Nevertheless, if the loading may not grow over a given limit L_{max} , then the largest N for which the BIST architecture is feasible is $N_{max} = 2 \cdot L_{max} / k$ (note that L_{max} depends on the BIST clock frequency). When N grows over this limit, we divide the FPGA in four quadrants, such that it is feasible to implement the BIST architecture separately in each quadrant. For example, if $N_{max}=8$, then an 16×16 FPGA will be divided into four 8×8 FPGAs. All quadrants are tested concurrently. However, the scan chains of each quadrant will be connected into a single scan chain, so that the result retrieval time grows linearly with the size of the FPGA.

We emphasize that increasing N does not affect the number of test sessions, which is always two. The number of test configurations (phases) depends only on the structure and the modes of operation of the PLB, and it is independent of N (the dependence shown in Figure 6 of [20] for the BIST method is a result of incorrect assumptions). Since all BUTs are tested in parallel, the BIST execution time is also independent of the size of the FPGA. In addition to the time for scanning out the results, the only test-time dependence on the size of the array is the reconfiguration time for each test phase. This is inherent in all the currently available FPGAs, for which the configuration loading is a serial process. The time of the method described in [21] does not depend on N , but this is based on a parallel loading mechanism that is not featured in existing FPGAs.

3. Boundary Scan Access

Practically all recently developed FPGAs, such as [5][26][43], feature a boundary-scan interface con-

trolled by a Test Access Port (TAP) [35], that can be also used for reconfiguration. This allows FPGAs to be reconfigured and tested in-system without requiring additional I/O pins. The Lucent [26] and the Xilinx [43] architectures also provide user-defined access to the FPGA core, which we use to control the ORA scan chain. Altera FPGAs [5] do not feature user-defined TAP instructions, but the equivalent functionality may be implemented in test mode at the cost of several test-dedicated I/O pins.

Reconfiguring the FPGA with the BIST test phases, initiating the BIST sequence, and reading the BIST results (steps 1, 2, and 5 of each test phase) are performed using the TAP. Access to our BIST architecture requires the ability to control the *BIST Start/Reset* to initialize the TPGs and ORAs and start the BIST sequence, as well as the ability to read the *BIST Done* and *ORA Pass/Fail* results from the BIST circuitry. This access must be made within the confines of the typical FPGA boundary-scan circuitry architecture. For example, unlike boundary-scan cells for external interconnect testing, the *Shift/Capture* and *Update* control signals are not made available to the core of the FPGA by the TAP circuitry in most FPGAs. Instead, the only internal signals provided by the TAP for user-defined internal scan chains are *TCK*, *TDI*, a port to send data out on *TDO*, and an internal enable signal (*TEN*). *TEN* is active when the user-defined scan-chain instruction is decoded by the TAP controller and remains active until a different instruction is loaded into the TAP instruction register. Additional considerations in selecting a boundary-scan access method for final implementation include total test time, PLB overhead, routability, and diagnostic resolution. A detailed analysis of four methods for access to the FPGA BIST architecture via the boundary-scan interface was given in [13]. In this section, we describe the method selected for the final implementation of our BIST approach.

By integrating an internal scan chain in the PLBs containing the ORAs, we obtain an architecture that allows us to observe the results of the comparisons done by every ORA, without additional logic resources and with only local routing resources for the scan chain. By using the data-select MUX that is part of the flip-flop circuitry in the most PLBs, we can alternately use the flip-flop to latch mismatches in the ORA and to shift out the *Pass/Fail* results at the completion of the BIST sequence. As a result, we can create individual, independent ORAs with integrated scan registers in each ORA as illustrated in Figure 5. This provides a significant enhancement to diagnostic resolution as will be discussed in Section 5.

Working within the confines of the typical FPGA boundary scan architecture, we use the *TEN* signal

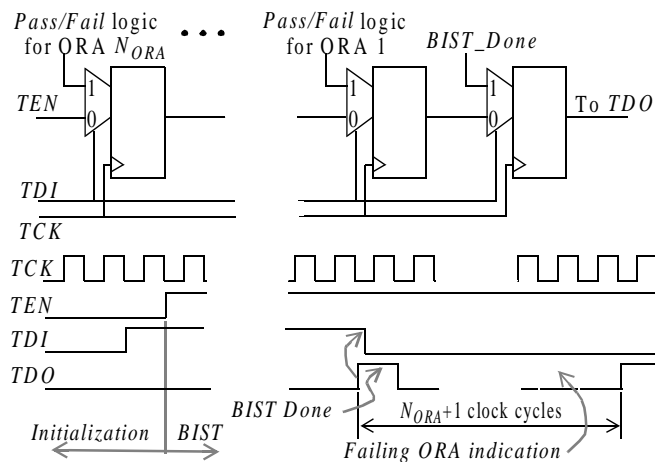


Figure 5. Results scan-chain and timing diagram

for the *BIST Start/Reset* functions. As a result, the FFs in the TPGs, ORAs, and BUTs are being reset until we load the user-defined scan register instruction, at which time the BIST sequence begins. This is accomplished by having *TEN* reset all FFs in the FPGA. The *TDI* input is used as the *Shift/Capture* control to the internal scan chain containing the BIST outputs (*BIST Done* and *Pass/Fail* indications) as illustrated in Figure 5. By connecting the *BIST Done* output to the last register in the internal scan chain, this signal is immediately observable on *TDO* by holding *TDI* at the *Capture* logic value (logic 1 in our implementation). As soon as *BIST Done* goes active, we begin shifting out the *Pass/Fail* indications by setting *TDI* to the *Shift* logic value (logic 0 in this case). $N_{ORA}+1$ clock cycles are needed to retrieve all the BIST results. By supplying a logic 1 (via the active *TEN* signal) to the scan input of the first scan FF in the chain, we are able to test the integrity of the internal scan chain as well. For example, a scan chain FF s-a-0 will be detected by the absence of the logic 1 at the end of the BIST results-shifting sequence. (Since a value of 1 represents an error, a FF s-a-1 will be immediately detected.)

Since all BIST operations involve the TAP controller, this is the first FPGA subcircuit that is tested. For this we use the “Tapdance” test sequence introduced in [8] or a subset of it. If the FPGA is tested with ATE, then we can use the entire sequence, which is a comprehensive functional test. For in-system test, we exclude any subsequence that involves FPGA I/O pins other than the four boundary-scan pins. For example, we skip testing the instructions that capture data in the boundary scan register (these instructions are usually tested together with the board interconnect by a separate board-level test). In this way, we obtain a complete test for all TAP operations that will be used by our BIST sequence. Although this subset is essentially a short external test, it relies only on the same four pins used by BIST, and can be regarded as an intrinsic part of the BIST sequence.

To summarize the sequencing of the FPGA test via the boundary scan interface, after the TAP controller test described above has passed, the test controller sends an instruction to the TAP controller to access the configuration memory and then downloads the configuration bits for the first BIST test phase. After configuration, the controller sends an instruction to access the user-defined scan register, which initiates the BIST sequence. The *TDI* input is held high until the *BIST Done* signal goes active indicating that the ORA *Pass/Fail* results are valid. Then *TDI* is set low and the *Pass/Fail* results are shifted out as illustrated in Figure 5. If a logic 1 appears at the end of the shift sequence (indicating the scan chain is fault-free), the test controller moves to the next BIST phase and repeats this sequence of operations. This process continues until all BIST test sessions and phases have been executed. All failures are recorded and subsequently analyzed for diagnosis. If the FPGA has been tested in system and found to be fault-free, then it is reconfigured to its normal mode of operation. If the diagnosis locates defective PLBs, the FPGA can be repaired by reconfiguration.

4. BIST-Based Fault Detection

In this section we show that our BIST approach achieves complete fault detection for single faulty PLBs, and practically complete fault detection for multiple faulty PLBs. A faulty PLB in a TPG or in an ORA may not produce an error if its fault does not affect the operation of the TPG or ORA. Thus we will rely on detection of PLB faults only when a faulty PLB is under test (configured as a BUT).

Claim 1: *Any single faulty PLB is guaranteed to be detected.*

Proof: When the faulty PLB is a BUT, it receives the correct pseudo-exhaustive patterns from a fault-free TPG in every one of its modes of operation. The outputs of the faulty BUT are compared with a fault-free BUT fed by a fault-free TPG, and the comparison and error latching are done by a fault-free ORA. (Since the scanning mechanism of the ORA result register is tested as part of every test phase, we can assume that errors propagate through a fault-free scan chain.) Thus the faulty PLB is detected. ■

The more difficult question is whether we can have multiple faulty PLBs that mask each other, so that together they escape detection. Although, in general, we cannot claim that any possible combination of faulty PLBs will be detected, we can identify many large classes of multiple faulty PLBs whose detection can be guaranteed. In the following, when we analyze the detectability of a group G of faulty PLBs, we implicitly assume that all the other PLBs are fault-free (in other words, G is not a subset of a larger group of faulty PLBs). G is detected when any of its PLBs is detected.

Claim 2: *Any group of faulty PLBs in the same row is guaranteed to be detected.*

Proof: Except the blocks in a TPG, the only interaction among PLBs located in different columns is provided by the scan register connecting the ORA result flip-flops. However, the scan operation of the ORA result register is also tested by the technique described in Section 3. Hence, faults in BUT and ORA PLBs in the same row cannot mask each other. Faulty PLBs in rows 1 or N may not be detected when they are part of a TPG (for example, when two faulty TPGs feeding pairs of BUTs compared by the same ORAs generate identical patterns), but they will be detected when configured as BUTs. ■

The next result deals with the detection of faulty PLBs residing in the same column, called the *faulty column*. A *middle row* refers to any row except 1 and N .

Claim 3: *Any group of faulty PLBs in the middle rows of the same column that has at least two adjacent fault-free PLBs, is guaranteed to be detected.*

Proof: Consider two adjacent fault-free PLBs that have a faulty neighbor PLB. For illustration, assume that P and Q in Figure 6a are fault-free, and that R is faulty (denoted by a black cell). Consider the test session in which Q and R are BUTs and P is an ORA. The fault(s) in R must be detected, because all test patterns are applied by two fault-free TPGs (in row 1 or N) to R and Q , and R is compared with a fault-free BUT (Q) by a fault-free ORA (P). Thus the presence of the two adjacent fault-free PLBs does not allow their faulty neighbor to be masked, and therefore any group of faulty PLBs in the middle rows of the same column is detected. ■

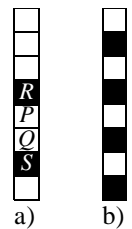


Figure 6.

Note that to escape detection, there must be at least $N/2$ faulty PLBs in the group, because otherwise the faulty column will have at least two adjacent fault-free PLBs. This is a very restrictive condition: for example, in a 20×20 FPGA, any faulty column with less than 10 faulty FPGA is guaranteed to be detected. Furthermore, the faulty PLBs configured as BUTs must produce identical output responses during every test phase. The reason Claim 3 restricts faulty PLBs to the middle rows of the faulty column is to guarantee that the TPGs in rows 1 and N are fault-free and thus generate all test patterns. Otherwise, it is theoretically possible that the group of faulty PLBs escape detection, even if the faulty column has two adjacent fault-free PLBs; for this to happen, all of the following conditions must be satisfied: 1) the faulty PLB in row 1 or N must change the patterns produced by one TPG, and, 2) the faulty TPG must skip all the patterns that detect every faulty BUT bordering two adjacent fault-free PLBs, and, 3) the fault-free BUTs must have

identical responses for every pair of different vectors produced by the two TPGs (otherwise errors will appear at all the ORAs in the fault-free columns). Clearly, this set of conditions is so restrictive that it is unlikely to ever occur in practice. Hence in practice, columns that include faulty PLBs in rows 1 or N will also be detected. (The above analysis shows that the claim made in [20] that BIST-based approaches “do not detect all double faulty PLBs” is incorrect.)

Figure 6b shows an example of a faulty column with $N=8$ where no pair of fault-free PLBs are adjacent. Consider the test session in which all 4 faulty PLBs are BUTs. In addition, assume that their faults are functionally equivalent, so that no mismatches will be produced. If these faults are not activated in the other test session when the faulty PLBs in the middle rows serve as ORAs, and the PLB in row 8 is part of a TPG, then this multiple fault will escape detection. However, this situation can be characterized as “pathological,” since it has an extremely low probability of occurrence: 1) half of the PLBs in the same column must be faulty, *and*, 2) these PLBs must reside in every other row *and*, 3) their faults must be functionally equivalent, *and*, 4) these faults must not be detected when these PLBs are configured as ORAs and TPG cell. The probability of occurrence is very low, because we need the AND of four conditions which are very unlikely by themselves.

Because masking cannot occur among faults in the middle rows of different columns, we can derive from Claim 3 the following more general result.

Claim 4: *Any group of faulty PLBs in the middle rows of the FPGA, such that every faulty column has at least two adjacent fault-free PLBs, is guaranteed to be detected.*

Like Claim 3, Claim 4 can also be extended in practice to cover faulty PLBs in rows 1 and N . Figure 7 illustrates an interesting group of 4 faulty PLBs that can escape detection if the following conditions are all satisfied: 1) PLBs X and Y have the same position in the first and the second TPG in row 1 *and*, 2) PLBs V and Z have the same position in the first and the second TPG in row 8, *and*, 3) the faults in X and Y are equivalent, *and*, 4) the faults in V and Z are equivalent, *and*, 5) the TPGs in row 1 skip all the patterns that detect faults in V and Z , *and*, 6) the TPGs in row 8 skip all the patterns that detect faults in X and Y . The faulty PLBs escape detection because in every test session, the patterns generated by the two TPGs are identical (so no mismatches are detected at any ORAs), and they miss the faults in the faulty BUTs. Clearly, this would be another pathological situation.

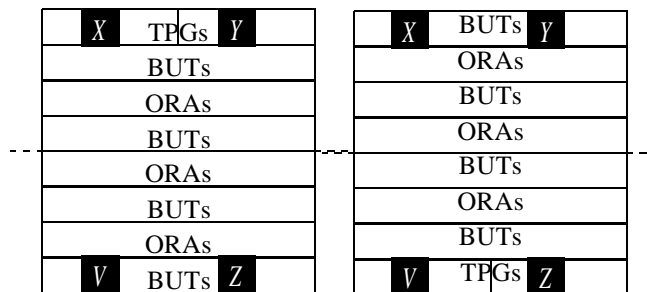


Figure 7. Pathological case that escapes fault detection

Although we cannot guarantee that the two test sessions illustrated in Figure 1 will detect any possible combination of faulty PLBs, it appears that the conditions that allow a group of faulty PLBs to escape detection are so restrictive, that they are very unlikely to occur in practice. Therefore we can conclude that, *in practice, any combination of faulty PLBs will be detected.*

But if needed, we can enhance the BIST sequence to guarantee the detection of any combination of faulty PLBs by adding the two test sessions shown in Figure 8. These are obtained by rotating the test sessions of Figure 7, so that the flow of test patterns is horizontal instead of vertical. If a group of faulty PLBs did escape detection in the first two test sessions, this was caused by masking among faulty PLBs in the same column. But the rotation is in effect interchanging the rows and the columns, so that it destroys the masking relations because the faulty PLBs interact now as if they are in the same row. Hence a group that escapes detection in the first two sessions is guaranteed to be detected in at least one of the additional two sessions.

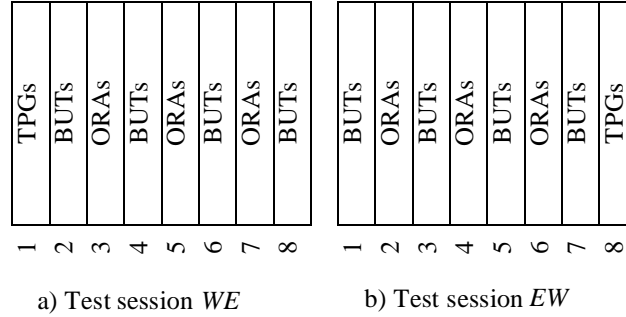


Figure 8. Additional (“horizontal”) test sessions

5. BIST-Based Diagnosis

In this section, we present the use of the BIST approach in diagnosing an FPGA that failed the tests provided by the two test sessions shown in Figure 1. In any fault location procedure, maximum diagnostic resolution is achieved when faults are isolated within an equivalence class containing all the faults that produce the observed response. If every equivalence class has only one fault, we say that the fault is *uniquely diagnosed*, that is, there is no other fault which can produce the same response. In our case a fault is one faulty PLB that may have any number of internal faults, and the response is obtained at the outputs of the ORAs. We begin by assuming a single faulty PLB in the FPGA, then we analyze the case of multiple faulty PLBs, and we also discuss locating faults inside a defective PLB.

5.1 Locating A Single Faulty PLB

Claim 5: *Any single faulty PLB is guaranteed to be uniquely diagnosed.*

Proof: First we analyze the case when a faulty PLB is detected only in the session when it is configured as a BUT. Note that most faulty BUTs will produce errors at two ORAs, except a faulty BUT in first two or the last two rows, which will produce an error in only one ORA. The column of the failing BUT is identified by the scan-chain position of the ORAs where errors are detected. Looking only at the faulty column, let O_i denote the ORA located in row i and B_j the BUT in row j . For example, a defective B_4 will cause errors at O_3 and O_5 in session NS , while a defective B_1 will be detected only at O_2 in session SN . The results of this analysis for an 8×8 FPGA are given in Table 1 under the heading “only BUT failures.” Errors at ORA outputs are marked by X. We can observe that the error pattern of every faulty row is unique.

Now we analyze the case when some faults in a PLB may also be detected when that PLB is configured as an ORA or a TPG. The results of this analysis are given in Table 1 under the heading “with

Table 1: Errors caused by a single faulty PLB

Faulty Row	Function Session NS	Function Sessio SN	only BUT failures						with potential ORA/TPG failures					
			Session NS			Session SN			Session NS			Session SN		
			O_3	O_5	O_7	O_2	O_4	O_6	O_3	O_5	O_7	O_2	O_4	O_6
1	TPG	BUT				X			(X X X)	X				
2	BUT	ORA	X						X			(X)		
3	ORA	BUT				X	X		(X)			X	X	
4	BUT	ORA	X	X					X	X			(X)	
5	ORA	BUT					X	X		(X)			X	X
6	BUT	ORA		X	X					X	X			(X)
7	ORA	BUT						X			(X)			X
8	BUT	TPG			X						X	(X	X	X)

potential ORA/TPG failures.” A fault in an ORA may cause an error only in that ORA when it reports a mismatch even though the compared pairs of output values agree. Thus in addition to the error at O_3 in session NS , a faulty B_2 may also cause an error at O_2 in session SN . This error is marked by “(X)” to denote a potential error. A fault in a TPG cell may cause the TPG to produce patterns different from those of a fault-free TPG, thus possibly generating mismatches in every comparator that observe the BUTs fed by the faulty TPG; in rows 1 and 8, we use “(X X X)” to denote a potential group of three errors. Thus, if all ORAs fed by the same TPG have errors in one session, we have an indication of faults in a TPG PLB, but we can ignore these errors and use the failures of the other test session to determine which PLB is faulty. Although every row in Table 1 now contains one potential error or one potential group of errors, it is easy to observe that the pattern of every faulty row is still different from all others. Therefore we can conclude that after the two BIST sessions we can accurately locate the row in which the faulty PLB resides. Combined with the column of the ORAs where errors are reported, this uniquely identifies the position of the faulty PLB.

■

This analysis is similar to the one done in [40], with the main difference being that the architecture of [40] allowed us only to locate the row containing the faulty PLB, while the new architecture shown in Figure 1 also provides us with the column of the faulty PLB. In the old architecture, the two additional test sessions illustrated in Figure 8, were required to locate the faulty column. These test sessions are no longer needed to locate a single faulty PLB in the new architecture.

5.2 Locating Multiple Faulty PLBs

The following results deal with the location of a group G of faulty PLBs that is detected in at least one test session. To uniquely diagnose G means to *identify all its faulty blocks such that no other group (including subsets of G) can produce the same result*. Although unique diagnosis is not always possible, there are many situations when it can be guaranteed.

Claim 6: *Any group of faulty PLBs in the same row is guaranteed to be uniquely diagnosed.*

Proof: In the session when all PLBs in the faulty row are under test, each BUT is observed at one or two ORAs (depending on the location of the faulty row) in the same column. Thus the ORAs where failures are observed are in different columns and do not interact. ■

Two PLBs in the same column are said to be *disjoint* if their faults cannot be observed at the same ORA. A group of PLBs in the same column is disjoint if every pair of PLBs in the group are disjoint. For

example, PLBs in rows 1, 4, and 8 of the same column form a disjoint group, but PLBs in rows 3 and 5 are not disjoint, since both of them are observed at the O_4 . Since faults in disjoint PLBs do not interact, we have the following results.

Claim 7: Any disjoint group of faulty PLBs in the same column is guaranteed to be uniquely diagnosed.

Claim 8: Any group of faulty PLBs in the FPGA such that the faulty PLBs in every column are disjoint, is guaranteed to be uniquely diagnosed.

Thus far for diagnosis of disjoint faulty PLBs, it has been sufficient to know the ORAs where errors are observed. But it is easy to see that this knowledge is not enough to diagnose non-disjoint faulty PLBs. For example, if two PLBs may be faulty and we obtain errors at O_3 and O_5 in session NS (see Table 1), we do not know whether B_2 is also faulty in addition to B_4 . However, in each test session we also record all failing phases, and we can use these data to achieve greater diagnostic resolution. If the sets of failing test phases obtained at O_3 and O_5 are different, this difference can be explained only by the faults in B_2 in addition to those obvious faults in B_4 .

Assuming that a faulty PLB is detected only when it is configured as a BUT, the set of failing phases obtained at O_i is given by:

$$FO_i = (FB_{i-1} \cup FB_{i+1}) - Feq_i \quad (1)$$

where FB_k is the set of failing phases of B_k , and Feq_i is the set of failing phases of both B_{i-1} and B_{i+1} that have identical responses (and thus do not cause mismatches at O_i). In Figure 9, the area of FO_i is marked by diagonal lines. Note that FO_i is empty (\emptyset) when both B_{i-1} and B_{i+1} are fault-free, or when the faults in B_{i-1} and B_{i+1} are equivalent (since then $FB_{i-1} = FB_{i+1} = Feq_i$). It is interesting to observe that there exists one situation when FO_i is the same, no matter if only one or both of the BUTs observed at O_i are faulty; this occurs if the two BUTs have the same sets of failing phases $FB_{i-1} = FB_{i+1}$ (\emptyset), but their faulty responses are ~~never the same~~ $Feq_i \neq \emptyset$.

Knowing the set of failing phases observed at O_i and the complete set of failing phases of one of the two faulty BUTs, we can determine the set of failing phases where the two BUTs have identical responses by

$$Feq_i = FB_{i-1} - FO_i = FB_{i+1} - FO_i \quad (2)$$

Based on FO_i and one of the two sets, we can also compute a lower bound on the other set by

$$FB_{i+1} \supseteq (FO_i - FB_{i-1}) \cup Feq_i \quad (3)$$

or by

$$FB_{i-1} \supseteq (FO_i - FB_{i+1}) \cup Feq_i \quad (4)$$

Note that (3) becomes an equality when FO_i and FB_{i-1} are disjoint:

$$FB_{i+1} = FO_i \cup Feq_i \quad (5)$$

This occurs when $FB_{i-1} \subseteq FB_{i+1}$ and $Feq_i = FB_{i-1}$

Next we outline a diagnostic procedure that, whenever possible, uniquely locates a group of faulty

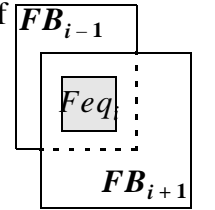


Figure 9.

PLBs in the same column and recognizes situations when unique diagnosis cannot be achieved. The procedure, called *MULTICELLO* (Multiple Faulty Cell Locator), relies on two assumptions which are valid in most practical situations:

A1: There are at most two interacting (non-disjoint) faulty BUTs having identical responses in the same failing phase.

A2: A faulty PLB works correctly when configured as an ORA.

Later we will discuss what happens when these assumptions are not true. The following results will be used by our diagnosis procedure.

Lemma 1: *A BUT observed by two ORAs that do not report failures in phase p does not fail in phase p .*

Proof: We denote “does not fail in phase p ” by \bar{p} (see Figure10). Assume, by contradiction, that B fails phase p . Let $B1$ be the BUT above $O1$ and $B2$ the BUT below $O2$. Since B fails p , but p is not reported by either $O1$ or $O2$, from equation (2) we conclude that both $B1$ and $B2$ must fail p with faulty responses identical to that of B . But then we would have three faulty BUTs with identical responses in p , and this would contradict assumption *A1*. Therefore B does not fail in phase p . ■

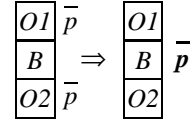


Figure 10

Lemma 2: *A BUT observed by two non-failing ORAs is fault-free.*

Proof: From Lemma1, a BUT observed by non-failing ORAs does not fail in any phase. ■

Lemma 3: *Let O be an ORA observing BUTs $B1$ and $B2$. If $B1$ does not fail in phase p and O does not report a failure in p , then $B2$ does not fail in phase p either.*

Proof: From equation (1) (see Figure 11). ■

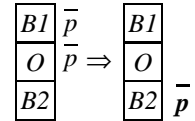


Figure 11.

Lemma 4: *Let O be an ORA observing BUTs $B1$ and $B2$. If $B1$ does not fail in phase p and O reports a failure in p , then $B2$ fails in phase p .*

Proof: From equation (1) and assumption *A2* (see Figure12). ■

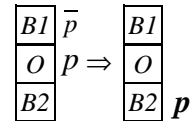


Figure 12

Lemma 5: *Let O be an ORA observing BUTs $B1$ and $B2$. If $B1$ fails in phase p and O does not report a failure in p , then $B2$ fails in phase p , and $B1$ and $B2$ have identical responses in phase p .*

Proof: From equation (2) (see Figure13). ■

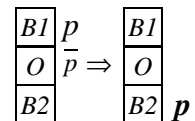


Figure 13

Next, we will present the *MULTICELLO* algorithm and at the same time we will illustrate its execution analyzing the responses obtained at the ORAs in one column in the *SN* test session of a 20×20 FPGA, where rows are numbered 1 to 20 with the TPGs in row 20 (see Figure14). The goal is to determine the set of failures for every BUT. The algorithm first identifies only non-failing phases for BUTs, then proceeds to determine the failing ones.

Procedure *MULTICELLO*:

- 1) Record ORA results and initialize the failures of every BUT in each phase as unknown.

This initial state is shown in Figure14 step1, where 0 and 1 entries for an ORA indicate, respectively, a passing and a failing result in the corresponding phase, and the empty cells denote unknown BUT failures. For example, *O8* reports failures in phases 1, 5, and 7.

- 2) In each column p , for every two consecutive ORAs with a 0 mark, enter a 0 for the BUT between them.

	1	2	3	4	5	6	7	8
B1								
O2	0	0	0	0	0	1	1	1
B3								
O4	0	0	0	0	0	1	0	0
B5								
O6	0	0	0	0	0	0	0	0
B7								
O8	1	0	0	0	1	0	1	0
B9								
O10	0	0	1	0	0	0	0	0
B11								
O12	1	1	1	0	1	0	1	0
B13								
O14	0	1	0	1	0	0	0	0
B15								
O16	0	0	0	1	0	0	0	0
B17								
O18	0	0	0	0	0	0	0	0
B19								

step 1

	1	2	3	4	5	6	7	8
B1								
O2	0	0	0	0	0	1	1	1
B3	0	0	0	0	0			
O4	0	0	0	0	0	1	0	0
B5	0	0	0	0	0	0	0	0
O6	0	0	0	0	0	0	0	0
B7		0	0	0	0	0	0	0
O8	1	0	0	0	1	0	1	0
B9		0	0	0	0	0	0	0
O10	0	0	1	0	0	0	0	0
B11			0	0	0			
O12	1	1	1	0	1	0	1	0
B13					0	0	0	0
O14	0	1	0	1	0	0	0	0
B15	0	0	0	0	0	0	0	0
O16	0	0	0	1	0	0	0	0
B17	0	0	0	0	0	0	0	0
O18	0	0	0	0	0	0	0	0
B19								

step 2

	1	2	3	4	5	6	7	8
B1	0	0	0	0	0			
O2	0	0	0	0	0	1	1	1
B3	0	0	0	0	0		0	0
O4	0	0	0	0	0	1	0	0
B5	0	0	0	0	0	0	0	0
O6	0	0	0	0	0	0	0	0
B7	0	0	0	0	0	0	0	0
O8	1	0	0	0	1	0	1	0
B9		0	0	0	0	0	0	0
O10	0	0	1	0	0	0	0	0
B11		0	0	0	0	0	0	0
O12	1	1	1	0	1	0	1	0
B13	0	0	0	0	0	0	0	0
O14	0	1	0	1	0	0	0	0
B15	0	0	0	0	0	0	0	0
O16	0	0	0	1	0	0	0	0
B17	0	0	0		0	0	0	0
O18	0	0	0	0	0	0	0	0
B19	0	0	0	0	0	0	0	0

step 3

	1	2	3	4	5	6	7	8
B1	0	0	0	0	0	1	1	
O2	0	0	0	0	0	1	1	1
B3	0	0	0	0	0		1	0
O4	0	0	0	0	0	1	0	0
B5	0	0	0	0	0	0	0	0
O6	0	0	0	0	0	0	0	0
B7	0	0	0	0	0	0	0	0
O8	1	0	0	0	1	0	1	0
B9	1	0	0	0	1	0	1	0
O10	0	0	1	0	0	0	0	0
B11	1	0	1	0	1	0	1	0
O12	1	1	1	0	1	0	1	0
B13	0	1	0	0	0	0	0	0
O14	0	1	0	1	0	0	0	0
B15	0	0	0	1	0	0	0	0
O16	0	0	0	1	0	0	0	0
B17	0	0	0		0	0	0	0
O18	0	0	0	0	0	0	0	0
B19	0	0	0	0	0	0	0	0

step 4

Figure 14 .Locating multiple faulty PLBs

This step applies Lemma1 and its results are shown in Figure14 step2 (new entries are shown in bold). We use the same 1 and 0 notation to respectively denote a BUT failure and a passing test.

3) In each column p , for every two adjacent 0 marks followed by an empty cell, enter a 0 in the empty cell.

This step applies Lemma3; the two adjacent 0 marks belong to a BUT and an ORA, and the empty cell is the other BUT observed by the same ORA. The results are shown in Figure14 step3. Note that $B5$ and $B7$ have already been identified as fault-free.

4) In each column p , for every adjacent 0 and 1 marks followed by an empty cell, enter a 1 in the empty cell.

This step applies both Lemma4 (when the 0 mark is for a BUT) and Lemma5 (when the 0 mark is for an ORA). The results are shown in Figure 14 step 4. We have identified 6 faulty PLBs - $B1$, $B3$, $B9$, $B11$, $B13$, and $B15$. Note that none of the failures of $B9$ is observed at $O10$, because $B11$ has identical responses in phases 1, 5, and 7. Also note that we have identified every failing phase for 5 out of the 6 faulty PLBs, but we cannot determine whether $B1$ also fails in phase 6.

5) Consistency check: If there is an ORA reporting a failure in phase p , while neither of the two BUTs observed by the ORA fails in p , 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.

This is not the case in our example, since $B17$ and $B19$ may have equivalent faults failing phase 4. To achieve unique diagnosis, we need to apply the horizontal sessions shown in Figure8.

End

The above example dealt with the results of only one test session. Under the assumption that a faulty PLB is detected only when configured as a BUT, the two sessions can be independently analyzed by the same procedure, and a different group of faulty BUTs may be identified in each session.

It is interesting to see how *MULTICELLO* handles the results produced by a single faulty PLB. For

example, assume that we obtain errors in several phases both at $O4$ and $O6$; then in step2 and step3 all BUTs except $B5$ are identified as fault-free, and then $B5$ is located as the source of the failing phases. Similarly, if errors are obtained only at $O2$, *MULTICELLO* will uniquely diagnose $B1$ as faulty.

However, there are two single faulty PLBs that *MULTICELLO* cannot uniquely diagnose. For example, if we obtain the same errors at $O2$ and $O4$, then all the BUTs except $B1$ and $B3$ are identified as fault-free, and $B3$ is diagnosed as faulty. But we cannot determine whether $B1$ is fault-free or faulty with the same set of failing phases as $B3$. (In the previous example, we could not determine whether $B1$ had a failure in phase 3.) Thus *MULTICELLO* correctly finds B_3 to be faulty, but $\{B_3\}$ and $\{B_1, B_3\}$ are indistinguishable. The source of this problem is that the first and the last BUT in a column are observed only at one ORA, while all the other BUTs are observed at two ORAs.

If we cannot achieve unique diagnosis, we can apply the horizontal test sessions shown in Figure8, so that the non-disjoint cells in the same column whose interaction made diagnosis difficult are no longer interacting. For the example above, $\{B_3\}$ and $\{B_1, B_3\}$ will be distinguished because $B1$ and $B3$ will be checked in separate rows. This represents an *adaptive diagnosis strategy*, where the tests to be applied next are determined based on the results obtained so far - here the additional test sessions are applied only if the initial two test sessions do not uniquely diagnose the faulty PLBs in the FPGA.

Figure 15 illustrates an interesting situation where even the addition of the horizontal sessions will not help in obtaining a unique diagnosis. Let us assume that in column 1, we obtain the set of failing phases FO at $O2$ and $O4$; as shown before, *MULTICELLO* identifies $B3$ as faulty and $B1$ as potentially faulty with the same set FO . If next we apply the horizontal test sessions, and for row 1, we obtain the same errors FO at $O2$ and $O4$, then the same PLB in the corner (row 1 and column 1, denoted by $R1C1$) is again potentially faulty. So the groups $\{R3C1, R1C3\}$ and $\{R3C1, R1C3, R1C1\}$ are indistinguishable. However, this is another pathological situation, since it requires these three specific PLBs to have the same sets of failing phases. There are only four such cases in the entire FPGA - one for each corner.

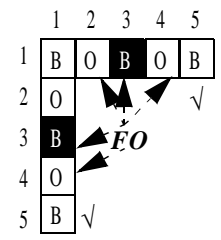


Figure 15

If *MULTICELLO* ends up detecting an inconsistency, this indicates either that the actual fault in the circuit is possibly an interconnect fault, or that some of the assumptions used are not valid. The most likely to be invalidated is the assumption about detecting a PLB only when configured as a BUT. Note that *MULTICELLO* will not do anything useful in a session where all ORAs in the same column have errors, since it needs two ORA without errors to execute step2. But these results are characteristic of a fault in a TPG cell, and diagnosis can still be successful in the other session where the TPG cells are BUTs. Although we can have a different diagnosis procedure to work under the assumption that faults in an ORA may modify its set of failures, instead we apply the horizontal test sessions and use *MULTICELLO* to process their results first.

In summary, we apply the two test sessions in Figure1, and we use *MULTICELLO* to diagnose the results of every column. If unique diagnosis is not achieved, we repeat the process with the horizontal test sessions in Figure 8. This adaptive strategy will achieve unique diagnosis for any group of faults encountered in practice.

Compared with the diagnosis algorithm for multiple faulty PLBs presented in [4], *MULTICELLO* is

simpler and relies on less restrictive assumptions.

5.3 Diagnosis Within A Faulty PLB

During each test session, the errors (failing ORA indications) are actually recorded for every test phase. This allows us to identify the failing mode(s) of operation of the faulty PLB and its faulty internal module(s). For example, consider the test phases developed for the ORCA 2C and 2CA series FPGA given in Table 2. The first 9 phases are used to test the various modes of operations of the PLBs in the ORCA 2C series, while the complete set of 14 test phases are used to test the ORCA 2CA series. Phases 1-4 test the LUT portion of the PLB, phases 5-9 test the FFs, and phases 10-14 test additional LUT modes of operation present only in ORCA 2CA series. For example, if only phase 10 fails, then we know that only the logic used exclusively to implement the multiplier is faulty. If only phases 5 and 7 fail, then the most likely cause is faulty connection(s) between the LUT and the flip-flops. Such accurate diagnosis is extremely useful in failure-mode analysis and yield improvement. This accuracy also provide the basis for allowing the reuse of a partially defective PLBs in fault-tolerant applications or adaptive computing applications [2]. For example, if only phases 5 through 9 fail, then the LUT is fault-free (since it passed exhaustive tests in each of its modes of operation), and this PLB with defective flip-flops can be safely used to implement any combinational logic function.

Additional diagnostic resolution can be obtained by constructing independent ORAs that compare the fewest possible BUT outputs per PLB flip-flop. As a result, a *Pass/Fail* indication is obtained for each pair of outputs being compared to give diagnostic information regarding which portion of the BUT is faulty. This ORA implementation is illustrated in Figure16, where each ORA flip-flop stores the result of only one comparison. In some FPGAs (such as the Xilinx 4000 and Vertex series [43] and ORCA 3C series [26]), independent ORAs can be implemented in one PLB to compare a single output from each of the two BUTs for maximum diagnostic resolution. Other FPGAs, such as the ORCA 2C series [26], are limited to comparing two pairs of BUT outputs due to the LUT architecture. When the test phases for the various modes of operation are coupled with the ‘fine-grain’ diagnostic resolution of the independent ORAs illus-

Table 2: Summary of BIST Phases for ORCA 2C and 2CA series Logic Blocks

Phase No.	Flip-Flop/Latch Modes & Options					LUT Mode	No. Outs
	FF/Latch	Set/Reset	Clock	Clk Enable	Flip-Flop Data In		
1	-	-	-	-	-	Asynchronous RAM	4
2	-	-	-	-	-	Adder/subtractor	5
3	-	-	-	-	-	5-variable MUX	4
4	-	-	-	-	-	5-variable XOR	4
5	Flip-Flop	async. reset	falling edge	active low	LUT output	Count up	5
6	Flip-Flop	async. set	falling edge	enabled	PLB input	Count up/down	5
7	Latch	sync. set	active low	active high	LUT output	Count down	5
8	Flip-Flop	sync. reset	rising edge	active low	PLB input	4-variable	4
9	Latch	-	active high	active low	dynamic select	4-variable	4
10	-	-	-	-	-	Multiplier	5
11	-	-	-	-	-	Greater/equal to Comp	5
12	-	-	-	-	-	Not equal to Comp	5
13	-	-	-	-	-	Synchronous RAM	4
14	-	-	-	-	-	Dual port RAM	4

trated in Figure 16, the faulty portion of a PLB module can be determined as well (for example, which one of the PLB flip-flops or LUTs are faulty). This accuracy in diagnosis was never achieved in any previous work.

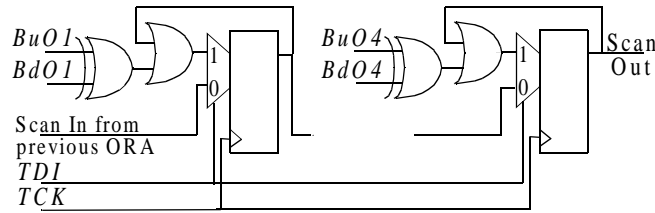


Figure 16 .ORA with greater diagnostic resolution

6. Experimental Results for ORCA FPGAs

In this section we present the results of the implementation of our BIST-based diagnostic approach using ORCA 2C15A FPGAs, and discuss our experience with testing and diagnosis of known defective FPGAs. The ORCA 2C15A has 400 PLBs in a 20×20 array and it requires the full set of 14 test phases summarized in Table 2 for each one of the two test sessions. Each test phase for the 2C15A requires 220,800 bits of memory to store the configuration for that test phase for a total storage requirement of 0.77 Mbytes for test configurations for both test sessions. The total FPGA test time, including all reconfigurations, is less than one second, using a 10 MHz boundary-scan clock.

We were provided with five 2C15A devices by Lucent Technologies Microelectronics Group in Allentown, PA. From manufacturing test results, two FPGAs were known to be fault-free and three were known to be defective. All three defective devices failed our test, and the two fault-free devices passed. Analyzing the results of the defective FPGAs, our PLB diagnosis procedure *MULTICELLO* reported inconsistencies for Chip1 and Chip2, and identified one faulty PLB in Chip3 (in row 3 and column 18).

Table 3: Summary of testing faulty FPGAs

Test applied	Chip1	Chip2	Chip3
Off-line PLB test (this paper)	FAIL	FAIL	FAIL
Off-line routing test [41]	FAIL	FAIL	PASS
On-line PLB test [2][3]	P&F	FAIL	FAIL

To validate these results, we retested the defective chips with the off-line BIST for programmable interconnect described in [41], and with the on-line BIST for PLBs (the Roving STARs approach) presented in [2][3]. No application was programmed in FPGAs for the on-line test. The on-line BIST tests each PLB twice, once in a vertical STAR and one in an horizontal STAR. Table 3 summarizes the results of all tests. The failures of the routing BIST in Chip1 and Chip2 show that the inconsistencies reported by *MULTICELLO* trying to locate defective PLBs in these devices are caused by the presence of interconnect faults. For Chip1, the on-line PLB test passed all vertical roving tests, but failed some of the horizontal roving tests. Since these tests involve the same PLBs but different routing resources, we concluded that Chip1 has only interconnect faults that also cause its off-line PLB test to fail. Chip2 fails every test and has both faulty PLBs and faulty interconnect. Chip3 passed the interconnect test, and the on-line diagnosis

method described in [3] identified the same faulty PLB as *MULTICELLO*.

The faulty PLB in Chip3 failed phases 5-9, and by using one additional configuration that separated between FFs and LUT, we were able to determine that the fault affects only the FFs and the LUT is fault-free [3]. This defective PLB has been successfully reused to implement combinational logic functions in a fault-tolerant application [11].

Since faulty FPGAs are difficult to obtain, we have also performed verification of our BIST approach using a fault emulator that injects faults in the FPGA by changing configuration bits just prior to download.

7. Conclusions

In this paper, we have described a BIST approach for programmable logic blocks in SRAM-based FPGAs. Our approach is applicable at any level of testing and, unlike conventional BIST, it does not introduce any area overhead or delay penalty. Every PLB is pseudoexhaustively tested in all its modes of operation, so the tests are practically complete for any logic fault model. We have shown that in practice, our method detects any combination of faulty PLBs. Our architecture facilitates the testing of all PLBs in only two test sessions, and the number of test configurations is also independent of the size of the FPGA. Its regular structure allows easy scalability with the size of the FPGA and algorithmic generation of placement and routing data for every test phase based only on the size of the array.

We also presented the first diagnosis algorithm that can accurately locate any single and most multiple faulty PLBs with maximum diagnostic resolution. The multiple faulty PLBs that cannot be diagnosed appear to be very restrictive situations unlikely to occur in practice. Our method can also identify defective subcircuits inside a PLB. This diagnostic information can then be used for yield enhancement in the manufacturing process or for repair strategies in fault-tolerant applications. We have successfully verified our approach with faults injected using a fault emulator and with actual defective FPGAs.

An open problem to be investigated in the future is diagnosis under a mixed-fault model, that is, locating faults in a chip that has both faulty PLBs and faulty interconnect.

Acknowledgments

The authors acknowledge the essential contributions to this project by Eric Lee (Lucent Technologies), Sajitha Wijesuriya (Lucent Technologies), and Carter Hamilton (Xilinx) during their graduate work in the VLSI-FPGA Design & Test Laboratory at the University of Kentucky. We also thank the reviewers for their very useful comments that helped us improve the paper. Finally, the authors acknowledge the support, assistance, and encouragement of C.T. Chen, Al Dunlop, and Carolyn Spivak of Lucent Technologies.

References

- [1] M. Abramovici, M. A. Breuer, and A. D. Friedman, "Digital Systems Testing and Testable Design," (revised printing) *IEEE Press*, 1995.
- [2] M. Abramovici, C. Stroud, S. Wijesuriya, C. Hamilton, and V. Verma, "Using Roving STARS for On-Line Testing and Diagnosis of FPGAs in Fault-Tolerant Applications," *Proc. IEEE International Test Conf.*, pp. 973-982, 1999.
- [3] M. Abramovici, C. Stroud, B. Skaggs, and J. Emmert, "Improving On-Line BIST-Based Diagnosis for Roving STARS", *Proc. IEEE International On-Line Test Workshop*, pp. 31-39, July 2000.

- [4] M. Abramovici and C. Stroud, "BIST-Based Detection and Diagnosis of Multiple Faults in FPGAs," *Proc. IEEE International Test Conf.*, October 2000.
- [5] Altera Corp., <http://www.altera.com/html/products/products.html>
- [6] A. Burress and P. Lala, "On-Line Testable Logic Design for FPGA Implementation", *Proc. International Test Conf.*, pp. 471-478, 1997.
- [7] B. Culbertson *et al.*, "Defect Tolerance on the Teramac Custom Computer," *Proc. IEEE Symp. on Field-Programmable Custom Computing Machines*, pp. 140-147, 1997.
- [8] T. A. Dahbura, M. U. Uyar, and C. W. Yau, "An Optimal Test Sequence for the JTAG/IEEE P1149.1 Test Access Port Controller," *Proc. IEEE International Test Conf.*, pp. 55-62, 1989.
- [9] D. Das and N.A. Touba, "A Low Cost Approach for Detecting, Locating, and Avoiding Interconnect Faults in FPGA-Based Reconfigurable Systems," *Proc. IEEE International Conf. on VLSI Design*, pp. 266-269, January 1999.
- [10] S. D'Angelo, C. Metra, G. Sechi, "Transient and Permanent Fault Diagnosis for FPGA-Based Systems," *IEEE International Symp. on Defect and Fault Tolerance in VLS*, pp. 330-338, November 1999.
- [11] J. Emmert, C. Stroud, B. Skaggs, and M. Abramovici, "Dynamic Fault Tolerance in FPGAs via Partial Reconfiguration," *Proc. 8th Annual IEEE Symp. on Field-Programmable Custom Computing Machines*, April 2000.
- [12] A. van de Goor, *Testing Semiconductor Memories Theory and Practice*, John Wiley and Sons, 1991.
- [13] C. Hamilton, G. Gibson, S. Wijesuriya, and C. Stroud, "Enhanced BIST-Based Diagnosis of FPGAs via Boundary Scan Access," *Proc. IEEE VLSITest Symp.*, pp. 413-418, May 1999
- [14] F. Hanchek and S. Dutt, "Methodologies for Tolerating Logic and Interconnect Faults in FPGAs," *IEEE Trans. on Computers*, pp. 15-33, Jan. 1998.
- [15] I.G. Harris and R. Tessier, "Interconnect Testing in Cluster-Based FPGA Architectures," *Proc. Design Automation Conf.*, June 2000.
- [16] F. Hatori *et al.*, "Introducing Redundancy in Field Programmable Gate Arrays," *Proc. IEEE Custom Integrated Circuits Conf.*, pp. 7.1.1-7.1.4, 1993.
- [17] W. K. Huang and F. Lombardi, "An Approach to Testing Programmable/Configurable Field Programmable Gate Arrays," *Proc. IEEE VLSITest Symp.*, pp. 450-455, 1996.
- [18] W. K. Huang, F. J. Meyer, N. Park, and F. Lombardi, "Testing Memory Modules in SRAM-based Configurable FPGAs," *IEEE International. Workshop on Memory Tech., Design and Testing*, August 1997
- [19] W. K. Huang, F. J. Meyer, X. Chen, and F. Lombardi, "Testing Configurable LUT-Based FPGAs," *IEEE Trans. on VLSI Systems*, Vol. 6, No. 2, pp. 276-283, June 1998.
- [20] W. K. Huang, F. J. Meyer, and F. Lombardi, "An Approach for Detecting Multiple Faulty FPGA Logic Blocks", *IEEE Trans. on Computers*, Vol. 49, No. 1, pp. 48-54, 2000.
- [21] T. Inoue, S. Miyazaki, and H. Fujiwara, "Universal Fault Diagnosis for Lookup Table FPGAs," *IEEE Design & Test of Computers*, Vol. 15, No. 1, pp. 39-44, Jan. 1998.
- [22] C. Jordan and W. P. Marnane, "Incoming Inspection of FPGAs," *Proc. European Test Conf.*, pp. 371-377, 1993.
- [23] J. L. Kelly and P. A. Ivey, "Defect Tolerant SRAM Based FPGAs," *Proc. International Conf. on Computer Design*, pp. 479-482, 1994.
- [24] V. Lakamraju and R. Tessier, "Tolerating Operational Faults in Cluster-based FPGAs," *Proc. ACM/SIGDA International Symp. on FPGAs*, pp. 187-194, Febr. 2000.
- [25] F. Lombardi, D. Ashen, X. Chen, and W. K. Huang, "Diagnosing Programmable Interconnect Systems for FPGAs," *Proc. ACM/SIGDA International Symp. on FPGAs*, pp. 100-106, Febr. 1996.

- [26] Lucent Technologies, Inc., <http://www.micro.lucnet.com/micro/fpga>
- [27] E. McCluskey, "Verification Testing - A Pseudoexhaustive Test Technique," *IEEE Trans. on Computers*, Vol. C-33, No. 6, pp. 541-546, June, 1984.
- [28] C. Metra, G. Mojoli, S. Pastore, D. Salvi, and G. Sechi, "Novel Technique for Testing FPGAs," *IEEE European Design and Test Conf.*, pp. 89-94, Febr., 1998.
- [29] J. Narasimhan *et al.*, "Yield Enhancement of Programmable ASIC Arrays by Reconfiguration of Circuit Placements," *IEEE Trans. on CAD*, Vol. 13, No. 8, pp. 976-986, August 1994.
- [30] M. Renovell, J. Figueras, and Y. Zorian, "Test of RAM-Based FPGA: Methodology and Application to Interconnects," *Proc. IEEE VLSI Test Symp.*, pp. 230-237, 1997.
- [31] M. Renovell, J. Portal, J. Figueras, and Y. Zorian, "Testing the Interconnect of RAM-Based FPGAs," *IEEE Design & Test of Computers*, Vol. 15, No. 1, pp. 45-50, Jan. 1998.
- [32] M. Renovell, J.M. Portal, J. Figueras and Y. Zorian, "SRAM-based FPGA: Testing the LUT/RAM Modules", *Proc. IEEE International Test Conf.*, pp. 1102-1111, 1998.
- [33] M. Renovell, J.M. Portal, J. Figueras and Y. Zorian, "SRAM-based FPGA: Testing the Embedded RAM Modules", *J. of Electronic Testing: Theory and Application (JETTA)*, Vol. 14, No. 1/2, pp. 159-167, Jan./Feb. 1999.
- [34] 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.
- [35] "Standard Test Access Port and Boundary-Scan Architecture," *IEEE Standard P1149.1-1990*, May 1990.
- [36] A. Steininger and C. Scherrer, "On the Necessity of On-Line BIST in Safety-Critical Applications," *Proc. 29th Fault-Tolerant Computing Symp.*, pp.208-215, 1999
- [37] C. Stroud, P. Chen, S. Konala, and M. Abramovici, "Evaluation of FPGA Resources for Built-In Self-Test of Programmable Logic Blocks," *Proc. ACM/SIGDA International Symp. on FPGAs*, pp. 107-113, 1996.
- [38] C. Stroud, S. Konala, P. Chen, and M. Abramovici, "Built-In Self-Test for Programmable Logic Blocks in FPGAs (Finally, A Free Lunch: BIST Without Overhead!)", *Proc. IEEE VLSI Test Symp.*, pp. 387-392, 1996.
- [39] C. Stroud, E. Lee, S. Konala, and M. Abramovici, "Using ILA Testing for BIST in FPGAs", *Proc. IEEE International Test Conf.*, pp. 68-75, 1996.
- [40] C. Stroud, E. Lee, and M. Abramovici, "BIST-based Diagnostics for FPGA Logic Blocks," *Proc. IEEE International Test Conf.*, pp. 539-547, 1997.
- [41] C. Stroud, S. Wijesuriya, C. Hamilton, and M. Abramovici, "Built-In Self-Test of FPGA Interconnect," *Proc. International Test Conf.*, pp. 404-411, 1998.
- [42] S.-J. Wang and T.-M. Tsai, "Test and Diagnosis of Faulty Logic Blocks in FPGAs," *Proc. IEEE International Conf. on Computer Aided Design*, pp. 722-727, 1997.
- [43] Xilinx, Inc., <http://www.xilinx.com/products>