Testing and Diagnosis of Interconnect Faults in Cluster-Based FPGA Architectures

I. G. Harris and R. Tessier

***Abstract*—As IC densities are increasing, cluster-based field programmable gate arrays (FPGA) architectures are becoming the architecture of choice for major FPGA manufacturers. A cluster-based architecture is one in which several logic blocks are grouped together into a coarse-grained logic block. While the high-density local interconnect often found within clusters serves to improve FPGA utilization, it also greatly complicates the FPGA interconnect testing problem. To address this issue, we have developed a hierarchical approach to define a set of FPGA configurations which enable interconnect fault detection and diagnosis. This technique enables the detection of bridging faults involving intracluster interconnect and extracluster interconnect. The hierarchical structure of a cluster-based tile is exploited to define intracluster configurations separately from extracluster configurations, thereby improving the efficiency of the configuration definition process. The cornerstone of this work is the concise expression of the detectability conditions of each fault and the distinguishability conditions of each fault pair. By guaranteeing that both intracluster and extracluster configurations have several test transparency properties, hierarchical fault detectability is ensured.**

***Index Terms*—BIST, design for testability, interconnect, testing.**

I. Introduction

Over the past decade field programmable gate arrays (FPGAs) have become invaluable components in many facets of digital design. As a result of increased integration, FPGA devices are now used across a wide assortment of fault tolerant and mission critical digital platforms. This application-level diversity has necessitated increased interest in FPGA tests so that faulty components can be quickly identified and recovered. Given the large range of applications and programmable configurations each FPGA device may support, an FPGA test can be substantially more complex than application-specified integated circuit (ASIC) test, providing motivation for new efficient testing techniques. Information regarding defect location is particularly important in today's test environment since new techniques [1] have been developed that can reconfigure FPGAs to avoid faults. To operate effectively, these approaches require that the specific location of the fault be clearly identified.

The reconfigurability of FPGAs plays an important role in reducing on-chip testing hardware relative to ASICs. While ASIC discrete Fourier transform (DFT) approaches require the modification of circuit functionality to perform test, FPGA test hardware can be swapped out of the device once verification is complete. Reconfigurability does incur other test costs, including increased test generation complexity and increased test application time. Unlike ASICs, which require a single configuration for fault detection, FPGAs require multiple configurations to test an assortment of switch settings. In general, fault coverage is directly related to the number and scope of test configurations that are created. The fault coverage issue has been further complicated in recent years by the introduction of FPGA devices [2], [3] with millions of programmable switch points. This device capacity growth strongly suggests the need for a hierarchical and incremental approach to FPGA test and diagnosis. To support this need, contemporary FPGA devices now allow for rapid partial device reconfiguration [4], [3].

Our fault test and diagnosis approach is driven by recent advances and improvements in FPGA architecture. To take advantage of circuit locality, several FPGA companies [3], [5] have recently introduced *cluster-based* architectures [6]. These architectures group numerous primitive logic components, such as flip–flops and lookup tables, into coarse-grained logic clusters. To simplify device mapping, clusters exhibit a high degree of internal connectivity including the feedback of cluster logic outputs to cluster inputs without the need for re-entry into sparse global interconnect. The richness of the internal interconnect complicates testing by providing a large range of potential interconnect patterns. Since pad area increases at a slower rate than internal logic, external access to internal test points becomes increasingly difficult as device sizes scale. As a result, novel testing approaches are needed to address and effectively test densely interconnected cluster-based architectures.

In this paper, an FPGA fault test and diagnosis approach is described that performs built-in self-test (BIST) on a cluster-based FPGA device. During the testing process, a portion of the FPGA is configured as test generation and response circuitry for a cluster under test. As individual logic clusters and surrounding routing resources are verified, they subsequently may be used to perform testing on remaining, untested clusters. To demonstrate the approach, we present a technique to generate FPGA test configurations to detect and diagnose pairwise bridging interconnect faults. By restricting the programming of the lookup tables in the FPGA, we formulate the testing and diagnosis conditions as a set of straightforward functions of the inputs of each tile. The testing and diagnosis conditions for each fault and fault pair are used to direct the configuration process. By exploring the hierarchy inherent in the structure of cluster-based devices, our approach partitions the test configuration definition process to greatly improve the efficiency of the process. Since test configurations in our approach are replicated across the cluster array, the process of defining test configurations is independent of the size of the FPGA array.

The paper is organized as follows: Section III is a description of the cluster-based architecture whose testing we are targeting. Section IV presents the BIST approach. The diagnosis conditions for the targeted interconnect faults are presented in Section V. The algorithms for configuration definition are presented in Section VI. Experimental results are presented in Section VII, and conclusions are presented in Section VIII.

II. Previous Work

Research in FPGA testing has investigated a wide range of test architectures and techniques. The FPGA test problem has been divided by several researchers into the interconnect test problem [7]–[10] and the FPGA logic test problem [11], [12]. The limited number of I/O pads greatly reduces test access from off-chip. The pad limit has been overcome by several researchers by using a number of BIST techniques [13], [14], [10], [15] to reduce the need for pads. Some portion of the FPGA hardware is configured as test generation and response analysis circuitry which is used to test the remainder of the FPGA. In order to test all of the FPGA logic, several configurations are required by these techniques to ensure that all FPGA logic is tested in some configuration. Several approaches to on-line fault detection have been introduced which implement BIST by exploiting unutilized FPGA logic and routing to implement modular redundancy [16], [15]. Previous work in

Manuscript received August 31, 2000; revised July 5, 2001. This work was supported in part by the National Science Foundation under Grant 0081343. This paper was recommended by Associate Editor C.-K. Cheng.

The authors are with the Department of Electrical and Computer Engineering, University of Massachusetts, Amherst MA 01003 USA (e-mail: harris@ecs.umass.edu).

Publisher Item Identifier 10.1109/TCAD.2002.804108.

0278-0070/02$17.00 © 2002 IEEE

test and diagnosis of the ORCA architecture [10], [15] is notable because this architecture can be classified as cluster-based.

Testing for delay faults in FPGA architectures has been investigated in previous work [17]–[20]. Some techniques target delay faults in the entire FPGA structure [19], [20], while other techniques specifically target delay only delay faults which impact a particular application [17], [18].

The need for external controllability and observability has also been reduced by using an iterative logic array (ILA) test architecture [11], [13], [14]. An ILA architecture composed of an array of identical cells allows the controllability and observability of each cell to be effectively accomplished through its neighboring cells. We have addressed the problem of defining a set of test configurations for cluster-based architectures previously in [21], and the diagnosis problem in [22].

III. Cluster-Based FPGAs

We assume an island-style FPGA architecture [23] which is composed of an array of identical tiles as shown in Fig. 1(a). Each tile is composed of a *cluster* [24] and surrounding interconnect. The interconnect structure of each tile is a set of lines which can be connected by a set of *programmable interconnect points* (PIP) which act as switches. A typical tile interconnect structure is shown in Fig. 1(b). The *switch matrix* shown in Fig. 1(c) is a commonly used structure in FPGA architectures which is composed of a set of lines entering each side. A PIP connects each line to one line on each side of the matrix. Each PIP in the switch matrix is seen as a dashed line in Fig. 1(c).

For the purposes of testing, it is necessary to distinguish the *tile I/O* from the *cluster I/O*. Cluster I/O are the input and output pins of the cluster, while tile I/O pins refer to the points at which a tile can communicate with a neighboring tile. The tile I/O pins include the endpoints of wire segments which can connect to a neighboring tile via a PIP.

We assume that each cluster is composed of a set of *basic logic elements* (BLE) [24], each of which is composed of a set of programmable *lookup tables* (LUT), multiplexers, and flip–flops. The most general assumption is that each BLE input can connect to the output of any other BLE and to any cluster input. The output of each BLE is assumed to be connected directly to a cluster output.

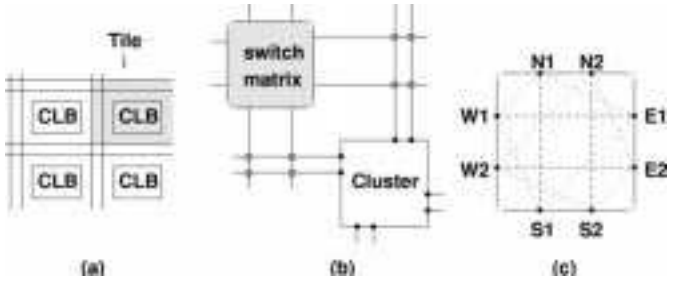

Fig. 1. FPGA structure: (a) tile array, (b) extracluster interconnect, and (c) switch matrix.

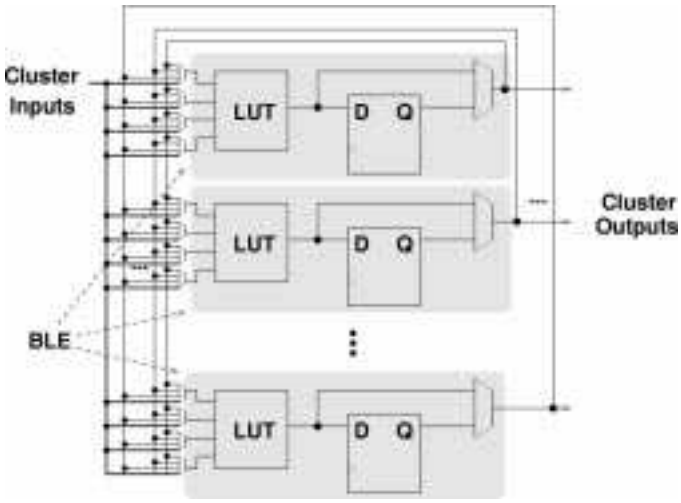

Fig. 2. FPGA cluster.

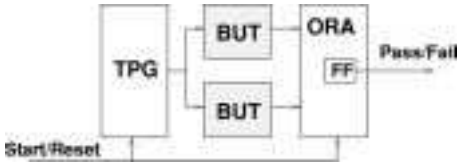

Fig. 3. BISTER test structure.

IV. FPGA Testing Methodology

We propose the use of a BIST strategy for the testing of an FPGA structure. BIST techniques, in general, are associated with high performance and area overhead incurred by on-chip test hardware. BIST overhead is not an issue for FPGA BIST because the test hardware is easily inserted and removed by reconfiguration. By embedding test logic inside the FPGA, BIST enables test access to internal components. This is particularly important for the testing of cluster-based FPGA structures which have higher localized interconnect density than other FPGAs.

In each configuration, FPGA circuitry dedicated as BIST logic will perform test generation and response analysis to test non-BIST FPGA circuitry. To accomplish BIST, we use the test structure presented in [10] in which the FPGA is configured as many independent BISTER's structures, shown in Fig. 3.

Each BISTER is composed of a test pattern generator (TPG), an output response analyzer (ORA), and two blocks under test (BUTs). The TPG is simply a counter which applies an exhaustive test sequence to the BUTs. Each BUT is a single tile in the FPGA which is being tested. The ORA is a comparator which sets the Pass/Fail flip-flop to "1" if the outputs of both BUTs do not agree. Each BISTER will be implemented as a rectangular block of tiles, and many BISTERs will be implemented on the FPGA to cover the tile array. The number of tiles in a BISTER will depend on the number of tiles needed to implement the TPG and ORA logic.

It is important to notice that the tiles which are dedicated to the TPG and ORA logic are not completely tested. In order to guarantee testing of all tiles, the FPGA will be reconfigured to shift the BISTERs across the entire array as shown in Fig. 4. Over the course of several reconfigurations, all tiles will be tested by acting as a BUT in a BISTER. Since the tiles adjacent to a BUT must implement either TPG or ORA logic, the perimeter tiles cannot be tested by simply shifting the BISTERs. In order to ensure that perimeter tiles are tested, the layout of the BISTER must be modified to use the I/O pads to access the tiles on the periphery.

The requirement of shifting the BISTER layouts over several reconfigurations causes the total test application time to be related to the area of the TPG/ORA logic. Since each block of TPG/ORA logic is used to test 2 tiles, each test configuration must be shifted $1 + \lceil A_t/2 \rceil$, where A_t is the number of tiles required to implement the TPG/ORA logic.

In addition to providing good test access, the use of this BIST strategy has several significant effects on the test configuration definition and test sequence definition problems. The BIST strategy decomposes the testing problem of the entire FPGA into many identical problems of a size which is fixed by the test requirements for

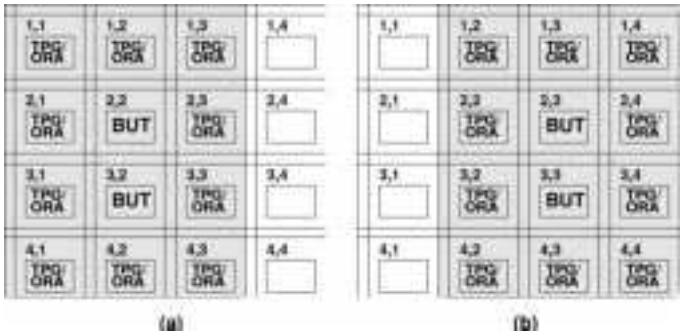

Fig. 4. Shifting BISTERs across FPGA array: (a) BISTER in lower left and (b) BISTER shifted right.

a single tile. Since the size of the smaller problem is fixed, the BIST approach is easily scalable to FPGA arrays of any size.

V. Interconnect Fault Detection and Diagnosis

Detection of interconnection faults in cluster-based architectures is a difficult problem because the high density of internal cluster interconnect makes test access difficult. We propose a formulation of the problem which includes the testing of *intracluster interconnect* which is internal to the cluster, as well as *extracluster interconnect* which surrounds each cluster. All pairs of lines are classified as either *connectable* if there is a PIP between them, and *nonconnectable* if there is no intervening PIP. We assume the possibility of two types of defects, a *short defect* which causes two lines to be crossed, and an *open defect* which causes a single line to be broken, or causes a connectable line pair to be unconnectable. Given the two classes of line pairs and the two defect types, we assume four fault classes which are previously presented in [7]. The interconnect faults which we target are subsets of bridging faults. The detection requirements of bridging faults in a non-FPGA context have been outlined in previous work [25], [26]. We summarize the four fault classes and the detection requirements for each class in terms of the controllability and observability of each line involved.

- **Permanent Connection (PC)**—A short on any pair of lines. Both affected lines must be separately controllable and at least one affected line must be observable. Also, any PIP between the two affected lines must be configured to be off.
- **Permanent Disconnection (PD)**—An open on any pair of connectable lines. Both affected lines must be controllable and observable. Also, the PIP between the two affected lines must be configured to be on.
- **Stuck-At 0 (SA0)**—A short between a line and ground (special case of a PC fault). The affected line must be controllable and observable.
- **Stuck-At 1 (SA1)**—A short between a line and power (special case of a PC fault). The affected line must be controllable and observable.

A. Detection and Diagnosis Requirements

In order to define FPGA configurations for testing and diagnosis, a clear definition of the testability and diagnosis requirements of each fault and fault pair is needed. Because the configuration defines the connectivity between segments, the test and diagnosis requirements must be expressed in terms of connectivity as well. We present requirements for the testability of each single fault and the differentiability of each fault pair. Detectability and differentiability indicate that some test pattern must exist to detect each fault and differentiate each fault pair. Detectability and differentiability requirements are independent of a specific test pattern, and they are used to define BUT test configurations before test pattern generation has been performed.

In the expressions presented in this section, we define the *control set* $C(s)$ of a segment s to be the set of tile I/O whose signal values determine the value of the segment. Because our approach configures all LUTs as four input XOR gates, the control set of a segment determines the function computed at segment s. We will define the *observe set* $O(s)$ of a segment to be the set of tile I/O to which a fault effect on segment s will be propagated. The observe set of a segment is the set of all tile I/O which are reachable from segment s in a configuration and are acting as tile outputs (are not being driven directly).

1) Fault Detection Conditions: In the testing approach proposed here, each segment value is restricted to be the exclusive–or of a subset of tile inputs. This is accomplished by configuring all LUTs to act as exclusive–or gates. Using this restriction, we can express the detectability of each fault as a function of the tile I/O which are reachable from the fault.

1. s_1**SA**v—

$$C(s_1) \neq \emptyset \cap O(s_1) \neq \emptyset.$$

This equation states that the faulty line segment s_1 must be both controllable by at least one tile input and observable by at least one tile output.

2. **PC**$(s_1,\ s_2)$—

$$C(s_1) \neq C(s_2) \cap C(s_1) \neq \emptyset \cap C(s_2) \neq \emptyset \cap (O(s_1) \neq \emptyset \cup O(s_2) \neq \emptyset).$$

This equation states that the faulty pair of segments must both be controllable $(C(s) \neq \emptyset)$, they must be separately controllable $(C(s_1) \neq C(s_2))$, and they must both be observable $(O(s_1) \neq \emptyset \cup O(s_2) \neq \emptyset)$. The PIP between the segments must be switched off.

3. **PD**$(s_1,\ s_2)$—

$$C(s_1) \neq \emptyset \cap O(s_2) \neq \emptyset$$

Assuming that s_2 is the floating segment, this equation states that the nonfloating segment must be controllable $(C(s_1) \neq \emptyset)$ and that the floating segment must be observable $(O(s_2) \neq \emptyset)$. The PIP between the segments must be switched on. Switching the PIP on implies that $C(s_1) = C(s_2)$.

2) Interconnect Fault Equivalence: The equivalence of faults limits the maximum achievable diagnostic resolution because equivalent faults cannot be differentiated. Fault equivalence in an FPGA is determined by the FPGA configuration, so faults which are equivalent in one configuration may not be equivalent in another. In order to achieve maximum diagnostic resolution, every pair of faults must be nonequivalent in at least one configuration.

We will define fault equivalence in terms of the connectivity between segments and tile I/O. Since the FPGA configuration determines connectivity, the process of configuration definition can ensure that all fault pairs are distinguishable. Two faults are said to be equivalent if their corresponding faulty machines produce the same output with all possible test patterns, at all outputs of the circuit. Since all LUTs act as exclusive–or gates in our approach, all fault effects are propagated to all outputs in the observe set of a faulty line. This implies that in order for two faults to be equivalent, the two segments at the fault location must have identical observe sets. The segments at the fault location must have identical control sets as well because fault effects must be generated by the same test patterns. We will refer to two segments as being *test equivalent* in a configuration if the segments have identical control sets and identical observe sets. Two test equivalent segments are indistinguishable during testing because they have the same value

under all input stimuli, and a fault effect on either segment will be observed at the same tile outputs. The equivalence of a fault pair depends on the test equivalence of the associated segments. We define the criteria for equivalence between all pairs of fault classes which may be equivalent.

1. s_1**SA**v/s_2**SA**v—The segments are test equivalent

$$C(s_1) = C(s_2) \cap O(s_1) = O(s_2).$$

This equation states that the two segments are controlled by the same set of tile inputs and observed by the same set of tile outputs.

2. **PD**(s_1, s_2)/**PD**(s_3, s_4)—s_1 and s_3 refer to the driver segments, and s_2 and s_4 refer to the floating segments

$$C(s_2) = C(s_4) \cap O(s_2) = O(s_4).$$

This equation states that each segment in a faulty segment pair must be test equivalent to a segment in the other faulty segment pair.

3. s_1**SA**v/**PD**(s_2, s_3)—This pair of faults may be equivalent if a segment which is not driven by a signal floats to a v value. In this case, the two faults are equivalent if the floating segment of the PD fault is test equivalent to the segment associated with the stuck-at v fault

$$C(s_1) = C(s_3) \cap O(s_1) = O(s_3).$$

This equation states that the segment with the stuck-at fault and the floating segment involved in the PD fault must be controlled by the same set of tile inputs and observed by the same set of tile outputs.

4. **PC**(s_1, s_2)/**PC**(s_3, s_4)—The pair of segments involved in one fault are test equivalent to the pair of segments involved in the other fault

$$\begin{aligned}&(C(s_1) = C(s_3) \cap O(s_1) = O(s_3) \cap C(s_2) = C(s_4) \cap \\ &O(s_2) = O(s_4)) \\ &\qquad \cup \\ &(C(s_1) = C(s_4) \cap O(s_1) = O(s_4) \cap C(s_2) = C(s_3) \cap \\ &O(s_2) = O(s_3))\end{aligned}$$

This equation states that each segment in a faulty segment pair (s_1, s_2) must be test equivalent to a segment in the other faulty segment pair (s_3, s_4).

VI. Test Configuration Definition

The goal of test configuration definition is to identify a set of configurations for the tiles acting as BUTs in a BISTER. The set of configurations must have the property that the fault detection requirements stated in Section V must be satisfied for all faults in at least one configuration. The size of the set of test configurations should be minimized to reduce test application time. The test configuration definition process is hierarchical, defining the intracluster configurations separately from the extracluster configurations. Test transparency constraints are placed on the intracluster and extracluster configurations to ensure hierarchical controllability and observability.

A. Intracluster Configurations

The intracluster configurations are defined to ensure that all intracluster interconnect faults are detectable in at least one configuration and to facilitate the testing of the extracluster interconnect. The cluster will be contained in the control and observe paths of many extracluster interconnect lines. The cluster must be configured to be *transparent* from a controllability and observability perspective. The cluster outputs are not identical to the cluster inputs, but the cluster outputs must have the following transparency properties with respect to the cluster inputs.

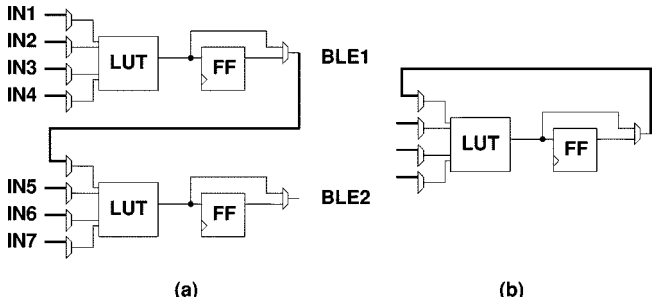

Fig. 5. Input multiplexer configurations: (a) BLE output function determined by input mux configurations and (b) illegal input mux configuration creating a self-loop.

1. A fault effect on a cluster input must propagate to at least one cluster output. This condition ensures the propagation of fault effects on extracluster which feed the cluster inputs.

2. The cluster outputs must be separately controllable. This condition ensures the controllability of the extracluster interconnect which is driven by the cluster outputs.

1) BLE Configurations: The observability of the cluster inputs and BLE output branches must be achieved by propagating fault effects through the BLEs to reach the cluster outputs. Also, the controllability of the BLE outputs must be achieved through the BLEs. The configuration of the BLEs is central to ensuring maximal controllability and observability inside the cluster. The configurations of components inside the BLEs are important to enable controllability of the BLE output lines, as well as observability of the cluster inputs lines and the BLE output branches.

Each BLE is composed of a LUT and a multiplexer, both of which must be configured. To maximize the controllability and observability through a BLE, we have chosen to configure each LUT to act as a four-input XOR gate. The XOR operation provides good controllability because the output value may be determined by controlling any single input. The XOR also provides good observability because a fault effect on any single input is guaranteed to propagate to its output. To simplify the interconnect testing process, the majority of test configurations bypass the flip–flop to drive the BLE output with the LUT output directly. This eliminates sequential behavior and ensures that the application of an exhaustive test pattern set is sufficient to detect all faults which are nonredundant in each configuration. In order to test the interconnect associated with the flip–flops, a single configuration is added in which all BLEs under test are configured so that their outputs are driven by the flip–flops. In this configuration all interconnect associated with flip–flops are made controllable and observable and therefore testable.

2) BLE Input Multiplexer Configurations: The configurations of the BLE input multiplexers (IMUX) affect both the controllability and observability of the cluster interconnect. The IMUXes determine controllability of BLE outputs by determining the function which defines the output of each BLE n. Because all LUTs are configured as XOR gates, each output BLE function is an XOR of a subset of cluster inputs as seen in Fig. 5(a). In Fig. 5(a), the input sources determined by the multiplexer configurations are labeled and shown in bold. Based on the multiplexer configurations, the BLE output functions are expressed as follows: $BLE1 = IN1 \oplus IN2 \oplus IN3 \oplus IN4$, $BLE1 = IN1 \oplus IN2 \oplus IN3 \oplus IN4 \oplus IN5 \oplus IN6 \oplus IN7$. Notice that the most general cluster-based architecture allows a multiplexer to be configured to create a loop as shown in Fig. 5(b), which would either create sequential activity (if the BLE output is clocked) or an asynchronous activity. Both of these possibilities would greatly complicate testing, so

we do not allow multiplexers to be configured to create a loop. This assumption matches the implementation of Xilinx Virtex [3] part which is a cluster-based architecture which does not contain interconnect to implement self-loops inside a cluster.

We have developed an algorithm to define the configuration of each IMUX in each overall FPGA configuration. We have identified the following IMUX configuration goals which must be satisfied by the algorithm. These properties are required to make all intracluster faults detectable in at least one configuration and to ensure the transparency of the cluster for the testing of extracluster faults.

• **All BLE outputs are separately controllable from each other, and from all cluster inputs**—This property ensures that each intracluster fault can be activated in each configuration and enables the activation of extracluster faults associated with extracluster lines driven by cluster outputs. Guaranteeing this property is accomplished by defining input multiplexer configurations so that each BLE output function is different, and is not dependent on a single cluster input.

• **Each input multiplexer is configured to select data from each of its inputs in at least one configuration**—This property ensures that all cluster input branches and BLE output branches are observable in at least one configuration.

• **There is a sensitized path from each cluster input stem to a cluster output in every configuration**—This property ensures the transparent propagation of extracluster fault effects through the cluster. This property is accomplished by configuring at least one input multiplexer to receive data from each cluster input in each configuration. Every cluster input stem can be associated with at least one BLE output whose value is dependent on that cluster input stem.

Algorithm 1 Intracluster Configuration Algorithm
```
label all intracluster faults as undetected
repeat
  repeat
    select a BLE which is not configured, b
    initialize IMUX configurations of b
    repeat
      enumerate next IMUX configuration
      compute BLE output function
    until BLE function is unique
  until all BLEs are configured
  identify detectable faults
until all faults are detectable in some configuration
```

The algorithm for intracluster test configuration definition is shown in Algorithm 1. The algorithm contains 3 main loops. The inner loop, defines the configuration of a single BLE by enumerating the configurations on all 4 of its input multiplexers until a satisfactory configuration is found. A set of BLE IMUX configurations is considered satisfactory if the resulting BLE output function is unique from the functions of all other BLEs, and is unique from all single cluster inputs. The middle loop invokes the inner loop with each BLE until all BLEs are configured to produce a complete cluster configuration. The outer loop invokes the middle loop to define a single configuration, and then evaluates the detection of intracluster bridging faults. The outer loop continues to invoke the middle loop until all intracluster faults are detected in at least one configuration.

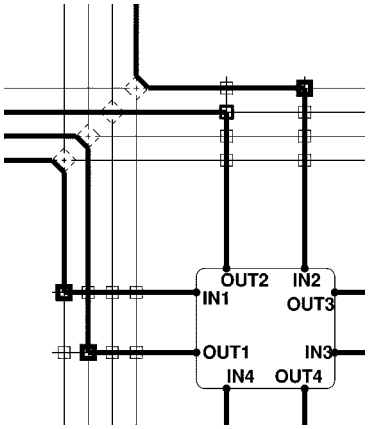

Fig. 6. Transparent extracluster configuration.

B. Extracluster Configurations

The extracluster configuration defines current flow paths through the extracluster interconnect. These current flow paths between tile input and output pins are used to control and observe each interconnect segment on the path. We model the extracluster configuration definition problem as a flow problem through an *interconnect graph*. Each node in the graph represents an extracluster interconnect segment, and each edge represents the existence of a PIP between two segments.

The goal of extracluster test configuration definition is to create flow paths between tile I/O nodes which allow the detection criteria of each fault to be satisfied in at least one configuration. In addition to enabling the detection of extracluster faults, the extracluster configuration must enable transparent controllability and observability of the embedded cluster. This goal is accomplished by creating flow paths from tile I/Os to every cluster input, and from every cluster output to tile I/Os, in every configuration. An example of a configuration which exhibits this type of test transparency is shown in Fig. 6. The bold lines indicate segments which are contained in flow paths, and the bold PIPs indicate which PIPs are switched on in the configuration in order to instantiate the paths. Each cluster input and output is directly connected to the edges of the tile via a set of flow paths. Notice that flow through the cluster does not impact the testability of extracluster interconnect because the cluster is transparent for controllability and observability purposes.

We present two different algorithms to generate extracluster test configurations which are used together to ensure the detection of all faults. The *fault independent* algorithm defines each configuration to maximize the fault coverage and distinguishability without targeting any faults specifically. The fault independent algorithm enables high fault coverage to be achieved with few configurations, but may fall short of 100% fault coverage for large examples. To achieve 100% fault coverage in cases where the fault independent algorithm is insufficient, the *fault specific* algorithm is used. The fault specific algorithm first selects an undetected fault and then defines a configuration which is guaranteed to detect the chosen fault. By using the fault specific algorithm after using the fault independent algorithm, 100% fault coverage is guaranteed.

Algorithm 2 Fault Independent Extracluster Algorithm
```
create interconnect graph
repeat
  label all nodes as untouched
  repeat
    select an untouched node n
```

TABLE I
EXPERIMENTAL RESULTS WITH A VARIETY OF CLUSTER SIZES

Clus. Prms.			Intra-Cluster				Extra-Cluster (Independent)			Extra-Cluster (Specific)		
N	I	SHFT	C	min	FCov	DiffCov	C	FCov	DiffCov	C	FCov	DiffCov
4	8	2	13	11	100.00%	100.00%	9	100.00%	99.97%	9	100.00%	99.98%
4	10	3	14	13	100.00%	100.00%	9	100.00%	99.97%	9	100.00%	99.98%
6	12	2	19	17	100.00%	100.00%	9	99.59%	99.98%	15	100.00%	99.98%
6	14	3	20	19	100.00%	100.00%	11	99.10%	99.97%	24	100.00%	99.98%
6	16	3	22	21	100.00%	100.00%	11	99.28%	99.97%	21	100.00%	99.98%
8	16	2	28	23	100.00%	100.00%	11	99.67%	99.99%	18	100.00%	99.99%
8	18	3	28	25	100.00%	100.00%	11	98.95%	99.95%	34	100.00%	99.98%
8	20	3	28	27	100.00%	100.00%	13	99.12%	99.97%	30	100.00%	99.98%

```
    identify an untouched path from n to
    a cluster I/O
    label all nodes on the path as
    touched
    identify an untouched path from n to
    a tile I/O
    label all nodes on the path as
    touched
  until paths connect all cluster I/O to
    tile I/O
  repeat
    select an untouched node n
    identify an untouched path from n to
    a tile I/O
    label all nodes on the path as
    touched
    identify an untouched path from n to
    a tile I/O
    label all nodes on the path as
    touched
  until no additional untouched paths
    can be created
    identify detectable faults and
    differentiable fault pairs
until fault coverage and diagnosability
do not improve
```

The fault-independent algorithm for extracluster test configuration definition is outlined in Algorithm 2. The two inner loops define a single test configuration by identifying a set of paths through the extracluster interconnect which must be activated. The first inner loop guarantees that the cluster I/O are directly controllable and observable from the tile I/O. The second inner loop serves to increase the number of extracluster interconnects which are controllable and observable. Each pass of the outer loop defines a single test configuration. The tasks, *select an untouched node* n, *identify an untouched path from* n *to a cluster I/O*, and *identify an untouched path from* n *to a tile I/O* are performed using several heuristics which target lines associated with faults which are undetected in the current configuration.

Algorithm 3 Fault Specific Extracluster Algorithm

```
repeat
  select an undetected fault f
  create paths to make f detectable
  complete configuration (fault
  independent algorithm)
  identify detectable faults and
  differentiable fault pairs
until all faults are detectable in some
configuration
```

The fault specific algorithm for extracluster test configuration definition is outlined in Algorithm 3. Each loop of this algorithm begins by selecting an undetected fault and defining paths through the interconnect to ensure that the selected fault is detectable. These paths are defined to satisfy the detection conditions stated in Section V with the shortest possible paths. Once the detection of the selected fault is ensured, the remainder of the configuration is completed using the fault independent algorithm to maximize coverage. The loop continues until all faults are detected.

VII. EXPERIMENTAL RESULTS

We have implemented the algorithms for test configuration definition and we have applied the algorithms to define test configurations for a range of cluster-based tiles of different sizes. In test results we assume that the cluster has the structure shown in Fig. 2 [6], with N BLEs and I cluster inputs. We assume that cluster inputs and outputs are equally distributed around the sides of the cluster. Each cluster I/O on the north face may connect to all horizontal tracks via a set of PIPs, and the same is true between cluster I/O on the west face and the vertical tracks. The cluster I/O on the east and south faces are assumed to connect directly to tracks in the neighboring tiles.

These results are summarized in Table I. The first three columns of Table I are the cluster parameters (*Clus. Prms*) which indicate the size and test properties of the cluster. The cluster parameters include the size of the cluster in terms of the number of cluster inputs, I, and the number of BLEs in a cluster, N. The third cluster parameter is $SHFT = 1 + \lceil A_t/2 \rceil$, where A_t is the number of tiles required to implement the TPG/ORA logic in a single BISTER. The value $1+\lceil A_t/2 \rceil$ is the number of times each configuration must be shifted in order to cover the testing of all tiles. The remainder of the columns in the table are divided into the results of *Intracluster* configuration definition, and two sets of extracluster configuration results. The first set of extracluster results labeled *Extracluster (Independent)* contain the results when only the fault independent extracluster algorithm is used. The set of results labeled *Extracluster (Specific)* contain the results when the fault specific extracluster algorithm is used to achieve 100% fault coverage after the fault independent algorithm has reached its coverage limit. The *C* columns contain the number of configurations defined. The *FCov* columns contain the fault coverage achieved, and the *DiffCov* columns contain the percent of fault pairs which are differentiated across all configurations. A *min* result is provided for intracluster results, indicating a theoretical lower bound on the number of intracluster configurations required. This lower bound is computed as the fanin of the input multiplexers ($I + N$), less 1 to account for the self-loop multiplexer input which we do not test.

The results in Table I show that high fault coverage and differentiation coverage are achieved in all cases. By using the fault specific extracluster configuration algorithm, 100% fault coverage can be guaranteed at the cost of an increased number of configurations.

VIII. Conclusion

We have presented a hierarchical technique to define test configurations for the detection and diagnosis of interconnect faults in cluster-based FPGA architectures. We have used the concept of test transparency to define configurations which enable test access to the high-density logic cluster embedded within each FPGA tile. We have demonstrated that this technique can be used to successfully define a small set of test configurations which allow the detection and diagnosis of nearly all targeted interconnect faults.

References

[1] V. Lakamraju and R. Tessier, "Tolerating operational faults in cluster-based FPGAs," in *8th Int. ACM/SIGDA Symp. Field Programmable Gate Arrays*, Feb. 2000.

[2] Altera Web Site [Online]. Available: http://www.altera.com/.

[3] "Virtex data sheet," Xilinx Corp., 2000.

[4] Atmel Corp., *Configurable Logic Design and Application Book*, 1997.

[5] Altera Corp., "Altera apex data sheet,", 2000.

[6] V. Betz, J. Rose, and A. Marquardt, *Architecture and CAD for Deep-Submicron FPGAs*. New York: Kluwer, 1999.

[7] M. Renovell, J. M. Portal, J. Figueras, and Y. Zorian, "Testing the interconnect of RAM-based FPGAs," *IEEE Design Test Comput.*, vol. 15, pp. 45–50, Jan.–Mar. 1998.

[8] M. Renovell and Y. Zorian, "Different experiments in test generation for xilinx FPGAs," in *Proc. Int. Test Conf.*, 2000, pp. 854–862.

[9] L. Zhao, D. M. H. Walker, and F. Lombardi, "Bridging fault detection in FPGA interconnects using i_{DDQ}," in *Int. Symp. Field Programmable Gate Arrays*, Feb. 1998, pp. 95–104.

[10] C. Stroud, S. Wijesuriya, C. Hamilton, and M. Abramovici, "Built-in self-test of FPGA interconnect," in *Int. Test Conf.*, Oct. 1998, pp. 404–441.

[11] M. Renovell, J. M. Portal, J. Figueras, and Y. Zorian, "SRAM-based FPGAs: Testing the LUT/RAM modules," in *Proc. Int. Test Conf.*, Oct. 1998, pp. 1102–1111.

[12] C. Metra, A. Pagano, and B. Ricco, "On-line testing of transient and crosstalk faults affecting interconnections of FPGA-implemented systems," in *Proc. Int. Test Conf.*, 2001, pp. 939–947.

[13] G. Gibson, L. Gray, and C. Stroud, "Boundary scan access of built-in self-test for field programmable gate arrays," in *Proc. IEEE Int. ASIC*, Sept. 1997, pp. 57–61.

[14] C. Stroud, E. Lee, and M. Abramovici, "BIST-based diagnostics of FPGA logic blocks," in *Proc. Int. Test Conf.*, Nov. 1997, pp. 539–547.

[15] M. Abramovici, C. Stroud, C. Hamilton, S. Wijesuriya, and V. Verma, "Using roving STAR's for on-line testing and diagnosis of FPGA's in fault-tolerant applications," in *Proc. Int. Test Conf.*, Sept. 1999.

[16] N. R. Shnidman, W. H. Mangione-Smith, and M. Potkonjak, "On-line fault detection for bus-based field programmable gate arrays," *IEEE Trans. VLSI Syst.*, vol. 6, pp. 656–666, Dec. 1998.

[17] I. G. Harris, P. Menon, and R. Tessier, "BIST-based delay-path testing in FPGA architectures," in *Proc. Int. Test Conf.*, 2001, pp. 932–938.

[18] J. G. Dastidar and N. A. Touba, "Improving diagnostic resolution of delay faults in FPGA's by exploiting reconfigurability," in *IEEE Symp. Defect Fault Tolerance*, 2001, pp. 215–220.

[19] A. Krasniewski, "Application-dependent testing of FPGA delay faults," in *Proc. EUROMICRO'99*, 1999, pp. 260–267.

[20] ——, "Enhancing detection of delay faults in FPGA-based circuits by transformations of LUT functions," in *Proc. IFAC Workshop Programmable Devices and Systems—PDS2000*, Feb. 2000, pp. 127–132.

[21] I. G. Harris and R. Tessier, "Interconnect testing in cluster-based FPGA architectures," in *Proc. Design Automation Conf.*, June 2000.

[22] ——, "Diagnosis of interconnect faults in cluster-based FPGA architectures," in *Proc. Int. Conf. Computer-Aided Design*, Nov. 2000.

[23] S. D. Brown, R. J. Francis, J. Rose, and Z. G. Vranesic, *Field-Programmable Gate Arrays*. New York: Kluwer, 1992.

[24] V. Betz and J. Rose, "Cluster-based logic blocks for FPGAs: Area-efficiency vs. input sharing and size," in *Proc. IEEE CICC*, 1997, pp. 551–554.

[25] M. J. Y. Williams and J. B. Angel, "Enhancing testability of large-scale integrated circuits via test points and additional logic," *IEEE Trans. Comput.*, vol. C-22, no. 1, pp. 46–60, Jan. 1973.

[26] M. Abramovici and P. R. Menon, "A practical approach to fault simulation and test generation for bridging faults," *IEEE Trans. Comput.*, vol. C-34, pp. 658–663, July 1985.