Diagnosis of Interconnect Faults in Cluster-Based FPGA Architectures

Ian Harris and Russell Tessier
Department of Electrical and Computer Engineering
University of Massachusetts at Amherst
E-mail: harris@ecs.umass.edu, tessier@ecs.umass.edu

Abstract—
Fault diagnosis has particular importance in the context of field programmable gate arrays (FPGAs) because faults can be avoided by reconfiguration at almost no real cost. Cluster-based FPGA architectures, in which several logic blocks are grouped together into a coarse-grained logic block, are rapidly becoming the architecture of choice for major FPGA manufacturers. The high density interconnect found within clusters greatly complicates the problem of FPGA diagnosis. We propose a technique for the testing and diagnosis of cluster-based FPGA architectures. We present a hierarchical approach to define a set of FPGA configurations in which each fault is detectable, and each fault pair is differentiable. The cornerstone of this work is the concise expression of the distinguishing conditions of each fault pair. Experimental results demonstrate that nearly 100% fault coverage and diagnostic resolution are achieved with a low number of test configurations.

I. Introduction

Over the past decade field programmable gate arrays (FPGAs) have become invaluable components in many facets of digital design. Information regarding defect location is particularly important in today's test environment since new techniques [8] 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 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, 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.

While previous researchers have investigated the test of FPGA architectures [10], [13], [11], [9], [3], our fault diagnosis approach is driven by recent advances and improvements in FPGA architecture. To take advantage of circuit locality, several FPGA companies [1] [6] have recently introduced *cluster-based* architectures [4]. These architectures group numerous primitive logic components, such as flip flops and look-up 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 diagnosis approach is described that performs built-in self test 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 diagnosis 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 II is a summary of the overall testing approach which was originally presented in [7]. The diagnosis conditions for the targeted interconnect faults are presented in Section III. Section IV presents the hierarchical configuration definition approach, and describes how diagnosis conditions are satisfied for fault pairs. Experimental results are presented in Section V, and conclusions are presented in Section VI.

II. Cluster-Based FPGA Test Methodology

We assume an island-style FPGA architecture [5] which is composed of an array of identical tiles. Each tile is composed of a *cluster* [4] and surrounding interconnect. The interconnect structure of each tile is a set of wire segments which can be connected by a set of switches referred to as *programmable interconnect points* (PIPs). A typical tile interconnect structure employs a *switch matrix* 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. 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.

The general model of a cluster, as presented in [4], is shown in Figure 1. We assume that each cluster is composed of a set of *basic logic elements* (BLE), 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.

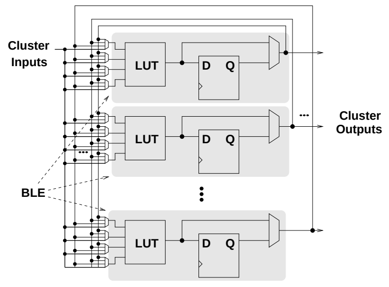

Fig. 1. FPGA Cluster

A. *FPGA Testing Methodology*

We propose the use of a built-in self-test (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 [11] in which the FPGA is configured as many independent BISTERs structures. Each BISTER is composed of a test pattern generator (TPG), an output response analyzer (ORA), and two blocks under test (BUTs). This 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 a single tile. Since the size of the smaller problem is fixed, the BIST approach is easily scalable to FPGA arrays of any size.

III. 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 *intra-cluster interconnect* which is internal to the cluster, as well as *extra-cluster interconnect* which surrounds each cluster. All pairs of lines are classified as either *connectable* if there is a PIP between them, and *non-connectable* 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 4 fault classes which are previously presented in [10]. The interconnect faults which we target are subsets of bridging faults whose detection requirements have been outlined in previous work [12], [2]. We summarize the 4 fault classes here.

- **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. *Fault Diagnosis*

In order to define FPGA configurations for testing and diagnosis, a clear definition of diagnosis requirements of each fault pair is needed. Because the configuration defines the connectivity between segments, the diagnosis requirements must be expressed in terms of connectivity as well. We present requirements for the differentiability of each fault pair. Differentiability indicates that some test pattern must exist to detect each fault and differentiate each fault pair. Differentiability requirements are independent of a specific test pattern and 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 4 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).

A.1 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 non-equivalent 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.

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

2. **PNC(**s_1**,** s_2**)/PNC(**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.

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

3. s_1 **SA**v**/PNC(**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 PNC fault is test equivalent to the segment associated with the stuck-at v fault.

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

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.

$$(C(s_1) = C(s_3) \cap O(s_1) = O(s_3) \cap C(s_2) = C(s_4) \cap O(s_2) = O(s_4)) \\ \bigcup \\ (C(s_1) = C(s_4) \cap O(s_1) = O(s_4) \cap C(s_2) = C(s_3) \cap O(s_2) = O(s_3))$$

IV. BUT 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 and differentiation conditions must be satisfied for all faults and fault pairs in at least one configuration. The number of test configurations should be minimized to reduce test application time. The test configuration definition process is hierarchical, defining the intra-cluster configurations separately from the extra-cluster configurations. Test transparency constraints are placed on the intra-cluster and extra-cluster configurations to ensure hierarchical controllability and observability.

A. Intra-Cluster Configurations

The intra-cluster configurations are defined to ensure that all intra-cluster interconnect faults are detectable in at least one configuration, and to facilitate the testing of the extra-cluster interconnect. The cluster will be contained in the control and observe paths of many extra-cluster 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.

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

A.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 4-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, we configure the multiplexers inside the BLEs to drive the BLE output with the LUT output directly, bypassing the flip-flop. This eliminates sequential behavior during testing and ensures that the application of an exhaustive test pattern set is sufficient to detect all faults which are non-redundant in each configuration.

A.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 Figure 2. In Figure 2, the input sources determined by the multiplexer configurations are labelled and shown in bold. Based on the multiplexer configurations, the BLE output functions are expressed as follows: $BLE1 = IN1 \oplus IN2 \oplus IN3 \oplus IN4$, $BLE1 = IN1 \oplus IN2 \oplus IN3 \oplus IN4 \oplus IN5 \oplus IN6 \oplus IN7$.

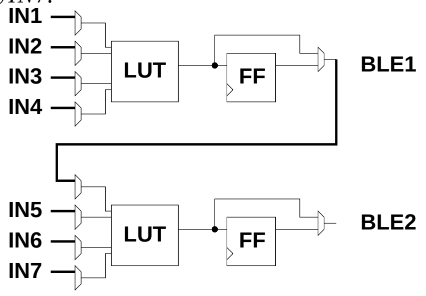

Fig. 2. BLE output function determined by input mux configurations.

We have developed an algorithm to define the configuration of each IMUX in each overall FPGA configuration. The test configurations must satisfy the detection and differentiation conditions for each fault and fault pair in at least one configuration. We have identified the following IMUX configuration requirements which ensure that detection and diagnosis goals are accomplished.

We propose criteria for the set of IMUX configurations which are satisfied by our algorithm. These criteria maximize the number of detected faults and differentiated fault pairs in each configuration. To maximize the number of differentiated fault pairs, **all CLB segments tested in a session must have distinct control sets.** The fault equivalence conditions described in Section III show that fault equivalence requires that lines associated with the faults must have identical control sets. By requiring distinct control sets, all detectable faults are also differentiable from all other faults. Several other criteria are used to maximize the detectability of interconnect faults and to guarantee that the intra-cluster logic is transparent to the extra-cluster testability [7]. The IMUX configuration algorithm enumerates the set of IMUX configurations in each CLB until all conditions are satisfied.

B. Extra-Cluster Configurations

The extra-cluster configuration defines current flow paths through the extra-cluster 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 extra-cluster configuration definition problem as a flow problem through an *interconnect graph*. Each node in the graph represents an extra-cluster interconnect segment, and each edge represents the existence of a PIP between two segments.

Fault equivalence can occur in extra-cluster interconnect between faults involving segment pairs which are connected electrically by an activated path. For example, if two segments s_1 and s_3 are connected by a PIP, and segments s_2 and s_4 are connected by a PIP, then the $PC(s_1, s_2)$ and $PC(s_3, s_4)$ faults are equivalent. Our algorithm minimizes fault equivalence by using heuristics to define connectivity paths for segments to avoid segments associated with faults which are not differentiated. In the case of the $PC(s_1, s_2)$ and $PC(s_3, s_4)$ faults, our algorithm would define connectivity paths for each of the four segments which are mutually exclusive. In this way, the four segments will not be electrically connected, and will not have identical control and observe sets.

Extra-cluster test configuration definition has two goals: 1) create flow paths between tile I/O nodes which allow the detection and differentiation criteria of each fault and fault pair to be satisfied, 2) enable the detection and differentiation of extra-cluster faults, the extra-cluster configuration must enable transparent controllability and observability of the embedded cluster. These goals are accomplished by our extra-cluster configuration algorithm by creating flow paths from tile I/Os to every cluster input, and from every cluster output to tile I/Os, in every configuration. Our extra-cluster configuration algorithm uses a modified shortest path algorithm to identify the necessary connectivity paths.

V. 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 Figure 1 [4], 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 two columns of Table I are the *Clus. Prms* which indicate the size of the cluster in terms of the number of cluster inputs, I, and the number of BLEs in a cluster, N. The remainder of the columns in the table are divided into the results of *Intra-Cluster* configuration definition, and *Extra-Cluster* configuration definition. The number of configurations defined by our algorithms are presented in the *Confs* column for both intra-cluster and extra-cluster configuration definition. A *min* result is provided for intra-cluster results, indicating a theoretical lower bound on the number of intra-cluster 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 percent of extra-cluster bridging faults detected across all configurations is shown in column *FCov*. The percent of extra-cluster fault pairs which are differentiated across all configurations are shown in column *DiffCov*. Fault coverage and differentiation coverage results are not shown for intra-cluster faults because both results were 100% in all cases.

The results in Table I show that testing the intra-cluster interconnect is the bottleneck in the number of configurations required. This is expected because the ratio of intra-cluster interconnect segments to cluster I/O pins is much higher than the 1:1 ratio between extra-cluster interconnect and the tile I/O. Notice that the fault differentiation percentage is sometimes higher than the fault coverage because two faults can be differentiated when one fault is detected and the other is not.

Clus. Prms.		Intra-Cluster		Extra-Cluster		
N	I	Confs	min	Confs	FCov	DiffCov
4	8	13	11	9	100.00%	99.97%
4	10	14	13	9	100.00%	99.97%
6	12	19	17	9	99.59%	99.98%
6	14	20	19	11	99.10%	99.97%
6	16	22	21	11	99.28%	99.97%
8	16	28	23	11	99.67%	99.99%
8	18	28	25	11	98.95%	99.95%
8	20	28	27	13	99.12%	99.97%

TABLE I
Experimental results with a variety of cluster sizes

VI. Conclusions

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] Virtex data sheet. *Xilinx Corporation*, 2000.

[2] M. Abramovici and P. R. Menon. A practical approach to fault simulation and test generation for bridging faults. *IEEE Transactions on Computers*, C-34(7):658–663, July 1985.

[3] M. Abramovici, C. Stroud, C. Hamilton, S. Wijesuriya, and V. Verma. Using roving STARs for on-line testing and diagnosis of FPGAs in fault-tolerant applications. In *International Test Conference*, September 1999.

[4] V. Betz, J. Rose, and A. Marquardt. *Architecture and CAD for Deep-Submicron FPGAs*. Kluwer Academic Publishers, 1999.

[5] S. D. Brown, R. J. Francis, J. Rose, and Z. G. Vranesic. *Field-Programmable Gate Arrays*. Kluwer Academic Publishers, 1992.

[6] Altera Corporation. *Altera Apex Data Sheet*. 2000.

[7] I. G. Harris and R. Tessier. Interconnect testing in cluster-based FPGA architectures. In *Design Automation Conference*, June 2000.

[8] V. Lakamraju and R. Tessier. Tolerating operational faults in cluster-based FPGAs. In *8th International ACM/SIGDA Symposium on Field Programmable Gate Arrays*, February 2000.

[9] M. Renovell, J. M. Portal, J. Figueras, and Y. Zorian. SRAM-based FPGAs: Testing the LUT/RAM modules. In *International Test Conference*, pages 1102–1111, October 1998.

[10] M. Renovell, J. M. Portal, J. Figueras, and Y. Zorian. Testing the interconnect of RAM-based FPGAs. *IEEE Design & Test of Computers*, 15(1):45–50, January-March 1998.

[11] C. Stroud, S. Wijesuriya, C. Hamilton, and M. Abramovici. Built-in self-test of FPGA interconnect. In *International Test Conference*, pages 404–411, October 1998.

[12] M. J. Y. Williams and J. B. Angel. Enhancing testability of large-scale integrated circuits via test points and additional logic. *IEEE Transactions on Computers*, C-22(1):46–60, January 1973.

[13] L. Zhao, D. M. H. Walker, and F. Lombardi. Bridging fault detection in FPGA interconnects using i_{DDQ}. In *International Symposium on Field Programmable Gate Arrays*, pages 95–104, February 1998.