A Brief Introduction to Application-Dependent FPGA Testing

Jie Qin
Dept. of Electrical and Computer Engineering
200 Broun Hall, Auburn University, AL 36849-5201
Email: qinjie1@auburn.edu

ABSTRACT: This paper gives a brief introduction to the application-dependent FPGA testing. In comparison with the conventional application-independent FPGA testing whose objective is to test the resources of a FPGA chip as exhaustively as possible, application-dependent FPGA testing is only performed on the resources used by a specific application implemented on the FPGA.

The multi-configuration strategy proposed in [1] requires only three simple configurations to test the interconnects and logic blocks of an arbitrary application implemented on a FPGA chip. Besides, using this approach, a very fast diagnosis scheme can be realized without extra test configuration. The approach will be discussed in details in this paper.

1. INTRODUCTION

Field Programmable Gate Arrays (FPGAs) feature their ability to be configured in the field to implement an arbitrary desired function according to the real-time demands. This ability of FPGAs can help people to achieve a faster design cycle, lower development costs and a reduced time-to-market compared to conventional Application-Specific Integrated Circuits (ASICs). FPGAs, therefore, are widely used in many applications such as networking, storage systems, communication, and adaptive computing [2].

Testing FPGAs requires solutions different from those applicable to ASICs [3]. In literatures, there are several different FPGA testing strategies. The one discussed in [4] is based on configuring several application circuits and exercising them with test vectors developed specifically for each circuit and supplied by an external tester. The second strategy of external testing techniques exploits regular internal structure and reconfigurability of an FPGA to concurrently examine its individual components - configurable logic blocks (CLBs) and interconnects [5]-[7]. The third strategy of testing techniques for FPGA is based on the concept of Built-In Self-Test (BIST) [8]-[10]. Its general idea is to exercise the FPGA in a number of self-test sessions, so that during each session a selected part of a FPGA, which is configured into Test Pattern Generators (TPGs), is examined using the remaining portions of the FPGA, which is configured into Output Response Analyzers (ORAs).

Basically, the three approaches mentioned above is to test, as exhaustively as possible, all possible operation modes of CLBs, interconnects, and other FPGA components. By doing so, it is reasonable to believe that the device pass the test will correctly operate in the field regardless of functions implemented on it. Such a test strategy, which can be called *application-independent testing*, is normally employed by the chip manufacturer (therefore, it is also referred to as *manufacturing test*).

However, there are some problems associated with application-independent testing of FPGAs. One is its low inefficiency of detection of timing-related faults because it is impossible to test even a small fraction of all possible interconnection patterns that may occur in the user-defined configurations [3]. The other is the decreased yield of FPGA vendors. In fact, some of FPGA chips that do not pass the application-independent test may be still usable for some specific designs because the defects are located in some places which are not used by these designs [1].

Based on such facts, the concept of *application-dependent testing* was introduced into the FPGA testing and well studied and discussed in [1][3][11]-[15]. The key point of the application-dependent testing is that only the resources used by a specific configuration would be tested and thus the disadvantage associated with the application-independent FPGA testing can be avoided. By using this scheme to test FPGAs, not only will the time used to perform a test decreases greatly [11], but also FPGA vendors can benefit from the increased yield [1]. Because of the direct benefit obtained from it, the

application-dependent testing technology has been adopted by Xilinx [16].

This paper is to introduce one of the application-dependent FPGA testing techniques proposed in [1]. The rest of the paper is organized as follows. In the Section 2, some background information about application-dependent FPGA testing is presented. Section 3 gives a brief introduction of the Multi-Configuration Strategy (MCS) proposed in [1]. The interconnect testing and CLB testing of MCS are discussed in details in Section 4 and Section 5 respectively. Finally the section 6 concludes the paper.

2. BACKGROUND

A static RAM based FPGA is composed of a two-dimensional array of configurable logic blocks (CLBs), programmable interconnects, and programmable input/output blocks (IOBs). To realize a specific user-given application on a FPGA chip, a compiler software provided by the FPGA vendor is required to divide the application into several parts with each of them small enough to be fit in a CLB, implement each part in a CLB, and finally connect all used CLBs through a programmed interconnect network. Only if all the programmable resources of the FPGA chip used by the application configuration function correctly, the application can run well on the chip. In order to discuss the application-dependent FPGA testing, it is important to give some detailed information about the fault models widely used and studied in FPGA testing.

Bridging Fault: The fault represents a short between a group of signals. The logic value of the shorted net may be modeled as 1-dominant (OR bridge), 0-dominant (AND bridge), or indeterminate, depending upon the technology in which the circuit [17]. In FPGAs, bridging faults are the most common failure mode in interconnects [11]. The detection of bridging faults in an application-dependent interconnect network was studied in [11] and [12].

Stuck-at fault: The fault is modeled by assigning a fixed (0 or 1) value to a signal line in the circuit and its most popular form are the single stuck-at faults [17]. It is one of the significant failure modes happened in interconnects [11]. [1] and [11] studied the stuck-at faults happened in interconnects in an application-dependent FPGA chip and proposed some approaches to detect them.

Delay Fault: The fault causes the combinational delay of a circuit to exceed clock period [17]. Because the delays caused by interconnection can account for 70% of the FPGA clock cycle period and the programmable interconnects are the primary source of large variations in propagation delays, testing for delay faults in FPGAs should focus on excessive delays in the interconnect network [3]. The C-exhaustive testing (combinationally-exhausive testing) proposed in [3] aims at the detection of delay faults in the interconnect network.

Because a programmable FPGA chip, to some extent, can be configured in the field to realize an arbitrary function, FPGA testing can be easily performed through a way of functional testing, the functional defects of the programmable resources in a FPGA chip are also well used in FPGA testing.

Interconnect defect: this defect can be usually modeled by bridging faults or/and stuck-at faults. [1] and [12] studied each fault respectively happened in interconnects.

CLB defect: The CLB is the basic unit of a FPGA chip. It usually contains a built-in Look-Up Table (LUT), which can be configured to realize any possible combinational output for a given number of inputs. A faulty CLB can be detected through the functional test of the CLB.

IOB defect: Typical IOBs have several programmable features as well as associated routing resources to/from the core of the FPGA. If the IOBs are faulty, the information exchange with other components in the system may not be possible or reliable [18][19].

The multi-configuration scheme strategy, which will be detailedly talked about in the rest of the paper, can cover all the stuck-at faults in interconnects as well as CLB defects related to a user-given design. Some terms which would be used in the strategy are listed as follows.

All-0 pattern: an all “0” vector.

All-1 pattern: an all “1” vector.

Maximum Sequential Depth: the maximum number of lip-flops in the longest path from a PI to a PO.

Sequential Distance: the number of flip-flops between two points in a digital sequential circuit.

3. MULTI-CONFIGURATION STRATEGY

The Multi-Configuration Strategy (MCS) proposed in [1] requires only three test configurations to test the stuck-at faults in interconnects as well as CLB defects related to an arbitrary application implemented on a FPGA. Besides, using this approach, a very fast diagnosis scheme can be realized without extra test configuration. Basically, the MCS test can be divided into two independent parts:

1. Interconnect Testing: the testing of interconnects used by the user application is performed through two different test configurations. This test can one hundred percent cover the stuck-at faults which may happen in the used interconnects. Because these two test configurations keep all the interconnect configuration untouched, no extra placing and routing is required for test configuration generation.

2. CLB Testing: Although the testing of CLBs used by the user application needs only one test configuration, the generation of this test configuration needs to modify the interconnect configuration. This may cause some potential problem which will be discussed in details in Section 5.

4. INTERCONNECT TESTING

A. TEST CONFIGURATIONS

As mentioned in Section 3, the interconnect testing is performed through two test configurations, both of which preserve the entire original interconnect configuration. These two configurations are obtained through a pretty similar way.

In the first test configuration, all the built-in LUTs in the used CLBs are reconfigured to implement logic *AND* functions. Also all the flip-flops in the user application need to be reprogrammed to a preset value "1". Such a test configuration with an *all-1 pattern* test vector at primary inputs (PIs) can detect all stuck-at-0 faults which may happened in the original interconnect configuration. The first test configuration can be illustrated as the general model shown in Figure 1. Observing the model, we can find out that the POs would be an *all-1 pattern* no matter after how many clock cycles if the original interconnect configuration is fault-free. However, if there is a single or are multiple stuck-at-0 fault/faults in the original configuration, the faulty "0" will finally propagate to the POs of the logic cone reachable from the faulty site through the *AND* network and can be observed at a reachable PO at the clock cycle which is equal to the sequential distance of the fault site to that PO. Because it is impossible that a *sequential distance* is greater than the *maximum sequential depth* (C_{max}), it can be reached that the original interconnect configuration is fault-free if no "0" is observed during C_{max} clock cycles. If a "0" or multiple "0"s is/are observed at Pos during this period, the original interconnect configuration has some stuck-at-0 faults. Because all the fault effects can be observed at the POs, all single and multiple stuck-at-0 faults can be detected through this test configuration.

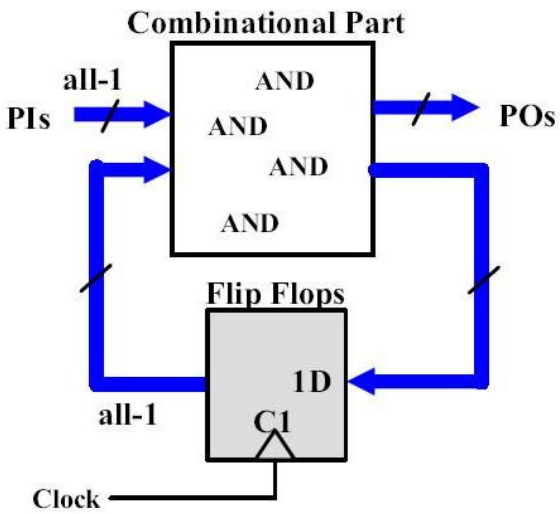

Figure 1 General Model of First Test Configuration [1]

In the second configuration, all the built-in LUTs in the used CLBs and are reconfigured to implement logic *OR* functions. At the same time, all the preset values of the flip-flops in the user application need to be reprogrammed to "0". In order to detect all the stuck-at-1 faults happened in the original interconnect configuration using this test configuration, an *all-0 pattern* test vector needs to be applied to the PIs correspondingly. The detection of stuck-at-1 faults is performed in a similar way in the first configuration. If no "1" is observed at the POs during C_{max} clock cycles, the

original interconnect configuration is fault-free, otherwise there must be a single or multiple stuck-at-1 fault happened.

B. FAULT DIAGNOSIS

The information extracted from these above two test configurations can also be used to locate the fault site where the stuck-at fault happens. Consider a LUT L and one of its input n_i. If there is a stuck-at fault on n_i, the fault will propagate to the output L and can be captured by the closest flip-flop connected to L at the next clock cycle. From now on, the faulty value will propagate through all the flip-flops at all possible path from L to POs and finally the fault can be observed at the fault-reachable POs. It is also apparently that the clock cycles at which the fault can be observed at all the fault-reachable POs depend on the minimum sequential distance between the fault site n_i and all these POs.

Based on the fault-reachable PO(s) and the failing clock cycles at which the fault is observed at each of these POs, the fault site can be roughly determined. A fault-reachable PO determines the faulty logic cone (LC) and the corresponding failing clock cycle specifies the sequential distance between the fault site and the PO. Each of fault-reachable POs and the corresponding failing clock cycle can give out a suspect set of fault sites. The rough location of the fault site can then finally be obtained from the intersection of these sets. The fault diagnosis algorithm is well illustrated from the pseudo code given in Figure 2.

```
Diagnosis Algorithm
1.For each failing output O_i do
2.   C ← Failing test clock cycle;
3.   Extract logic cone, LC, from O_i;
4.   S_i ← {nets with sequential
distance C from O_i in LC};
5.Suspect list ← ⋂_i S_i
```

Figure 2 Pseudo Code of Diagnosis Algorithm [1]

However, the diagnosis algorithm cannot locate the fault site precisely. Also the resolution of the diagnosis method, to a very great extent, is determined by the structure of the user-given application configuration under test. Consider such a case. There are multiple combinational paths from the fault site to the first flip-flop, *FF*, in the sequential path to a PO. Let N be the set of all nets in these paths. If there is no path from any net in N to a flip-flop other than *FF*, this method reports N as the suspect list and cannot reduce it any further. Although the resolution of this method is not as fine as we expected, it still can be considered effective due to the two facts listed below.

- Even in the worst case for the resolution of the method, the fault is localized to a very small portion of the design, which is absolutely adequate for many fault tolerant applications [20].
- This method is conservative, i.e., all faulty nets are a subset of the suspect list and no faulty net is escaped.

C. EXAMPLES

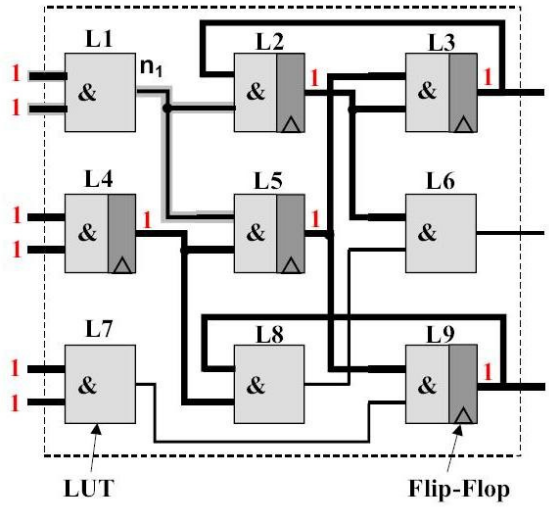

Figure 3 Example of Interconnect Testing[1]

Figure 3 shows an example of the first test configuration. Some CLBs (*L2, L3, L4, L5, L9*) in the original configuration use flip-flops as well as combinational functions, while the rest ones only performs combinational functions. The PIs and preset values of all flip-flops in this test configuration are "1", as shown in bold lines. Four test clock cycles are required to detect the possible stuck-at faults in original interconnect configuration because the maximum sequential

depth is four, which is shown in the longest path through *L4, L5, L3, L2 and L6*.

As an example, assume that there is a stuck-at-0 fault on the net n_1 in the configuration shown in Figure 3. Then the fault can be observed at the PO *L6* after the first clock cycle and at the POs *L3* and *L9* after the second clock cycle. From faulty PO *L6*, the suspect fault site set can be obtained as {*L1, L7*}. From PO *L3*, the fault site is included in the set {*L1, L4*}. From PO *L9*, the fault site should be in the set {*L1, L4*}. The intersection of the three suspect set gives the fault site at L1, which is equivalent to the net n_1 in the sequential distance sense.

5. CLB TESTING

As mentioned in Section 3, the CLB testing is performed using a test configuration which modifies the interconnect configuration and preserves the original CLB configuration. In this way, each CLB is tested in the same conditions which will be used in the original application configuration. At the same time, reprogramming the interconnect configuration makes the access to each CLB possible. Using this test configuration, this CLB testing covers a functional CLB defect model inclusive of stuck-at faults.

A. TEST CONFIGURATION

The key to CLB testing is to reprogram the interconnect network and make each used CLB controlled by the primary inputs (PIs) and monitored by a output response analyzer (ORA), which can be illustrated by Figure 4. The original application configuration and the test configuration are given in Figure 4-(a) and Figure 4-(b) respectively. It can found out that the CLB configuration, which is shown as gray boxes in Figure 4, is left untouched in the test configuration. Also a common bus directly connects the PIs to each CLB. Such a parallel connection configuration allows a concurrent test of the whole set of used CLBs. Because the number of the CLB is typically 4 in the Xilinx family, it is easy to generate all possible test vectors and drive them to these used CLBs. However, it is really an issue to monitor the output of each configured CLB under test because an application configuration usually uses much more CLBs than I/O pins. Therefore, an internal output response analyzer (ORA), which can generate a Go/NoGo signal according to the used CLBs' outputs, needs to be implemented in the test configuration. The MCS strategy chooses the classical XOR tree as the ORA's basic architecture and implements it with the unused CLBs in the original application configuration.

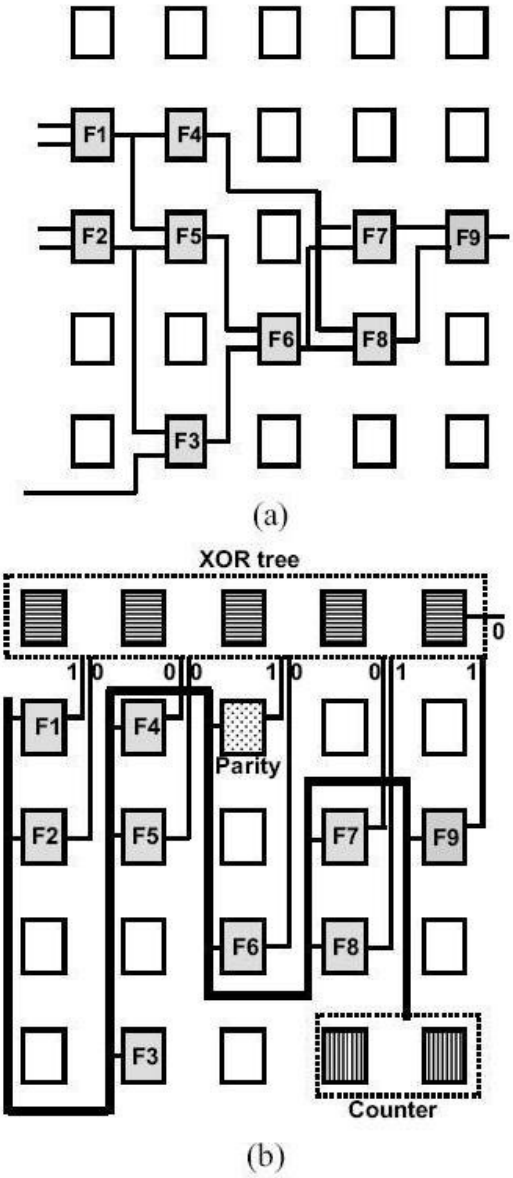

Figure 4 (a) Original Application Configuration (b) Modified Configuration [1]

B. BIST SCHEME

As shown in Figure 4, the entire test configuration is quite simple. Hence, a BIST version of the configuration can be realized with

some minor modification to the original test configuration. The BIST concept can be illustrated in Figure 5. Compared with the original configuration, the extra required components are a parity bit generator and an automatic test pattern generator (ATPG), which can be realized with a *N*-bit counter or a N-bit linear feedback shift register (LFSR) to generate all possible test vectors. With the parity bit, the output of the XOR-tree ORA is always "0" in a faulty-free circuit.

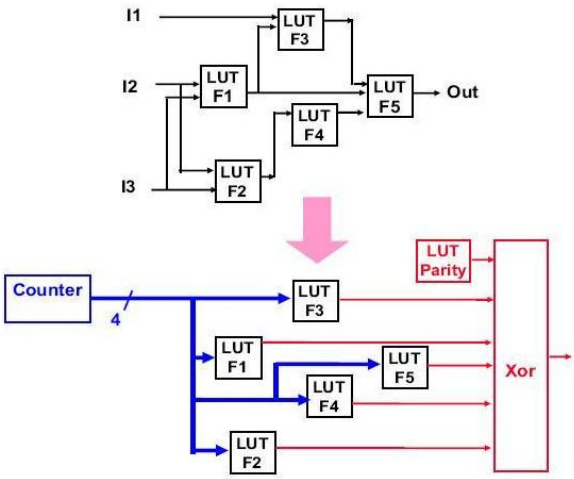

Figure 5 BIST Scheme of CLB Testing [1]

There is one issue related to the BIST configuration — routing congestion. Since all the test signal coming from the TPG must be routed to the inputs of all used CLBs， the resource of interconnects may be used up with no more to be allocated to the inputs of some CLBs. Under such a condition, the CLB testing cannot be performed in the way we talk about earlier. And the larger the application configuration under test is, the more serious this routing problem is. In order to solve this problem, multiple ATPGs can be implemented instead of single ATPG. Then the used CLBs can be partitioned into several parts with each part excited with one of the TPGs. The number and structures of the CLB partitions can be determined based on the routing constraints and the availability of spare logic resources.

C. Implementation of BIST Circuitry

Besides the original CLB configuration, extra components, such as the ORA and parity generator, need to be implemented in the CLB testing, so the extra circuitry added must be small enough to fit in the unused logic blocks, otherwise the CLB testing cannot be performed at all. Table 1 gives some implementation results of several benchmark circuits and their corresponding BIST circuitry.

Design	# of CLBs (initial)	# of CLBs (final)	# of CLBs (increased)
Table 5	177	183	*3.39%*
Table 3	207	217	4.83%
C5378	213	221	3.76%
C3540	227	230	1.32%

Table 1 BIST Implementation on Xilinx XC4000 [1]

From the Table 1, we can find out that the extra CLB overhead caused by BIST circuitry is small enough to fit into the FPGA chip compared with the CLBs used by the original application configuration. Even if the additional BIST circuitry cannot fit into the FPGA because of too much extra CLB overhead, or test vectors generated by the ATPG cannot be routed to the CLBs under test due to the routing constraints, the used logic can still be partitioned into two parts or more with each part tested by separate test configurations.

6. Conclusion

The basic difference between the application-dependent and the application-independent FPGA testing can be found out from the description in Section 4 and Section 5. In application-dependent FPGA testing, all programmable resources can be arbitrarily reconfigured to perform an exhaustive functional test of all the programmable resources including interconnects, CLBs, and IOBs in the FPGA chip. However, in application-dependent FPGA test configurations, some part of the original configuration must be preserved if this part needs to be test through this test configuration. Because of this problem, the availability of the programmable resources is a serious issue which needs to be considered in application-dependent FPGA testing.

The MCS strategy provides a simple way to perform the interconnect and CLB testing in the application-dependent context. But it still needs to be improved to achieve better performance. For

example, this strategy does not cover the bridging faults in interconnects, which are the most common failure mode in interconnects [11]. Also the resolution of the fault diagnosis algorithm in interconnect testing is greatly dependent on the structure of the original application configuration. The complexity of the CLB test configuration will increase in some worst case. All these problems need to be studied in the future research work in this area.

REFERENCES

[1] M. B. Tahoori, E. J. McCluskey, M. Renovell, P. Faure, "A Multi-Configuration Strategy for an Application Dependent Testing of FPGAs," Proc. VLSI Test Symp., 2004.

[2] M. B. Tahoori, "Application-Dependent Testing of FPGA Interconnects," Proc. Int'l Symp. On Defect and Fault Tolerance, 2003.

[3] A. Krasniewski, "Application-Dependent Testing of FPGA Delay Faults," Proc. 25th EUROMICRO Conf., vol. 1, pp. 260-267, 1999.

[4] C. Jordan, W. P. Marnane, "Incoming inspection of FPGAs", Proc. European Test Conf. pp. 371-377, 1993.

[5] W. K. Huang, F. J. Meyer, X.-T. Chen, F. Lombardi, "Testing Configurable LUT-Based FPGAs," IEEE Trans. on VLSI Systems, pp. 276-283, June 1998.

[6] T. Inoue, S. Miyazaki, H. Fujiwara, "Universal Fault Diagnosis for Lookup Table FPGAs", IEEE Design & Test of Computers, pp. 39-44, Jan.-March 1998.

[7] H. Shin, C. Kim, "Performance-Oriented Technology Mapping for LUT-Based FPGAs", IEEE Trans. on VLSI Systems, pp. 323-327, June 1995.

[8] M. Abramovici, C. Stroud, "BIST-Based Detection and Diagnosis of Multiple Faults in FPGAs," Proc. of Int'l Test Conf., 2000.

[9] C. Stroud, S. Wijesuriya, C. Hamilton, M. Abramovici, "Built-in self-test of FPGA interconnect," Proc. of Int'l Test Conf., pp. 404-411, 1998.

[10] X. Sun, J. Xu, B. Chan, P. Trouborst, "Novel Technique for Built-In-Self Test of FPGA Interconnects," Proc. of Int'l Test Conf., 2000.

[11] Das, N. A. Touba, "A Low Cost Approach for Detecting, Locating, and Avoiding Interconnect Faults in FPGA-Based Reconfigurable Systems,: Proc. of Int'l Conf. On VLSI Design, 1999.

[12] M. B. Tahoori, "Application-Specific Bridging Fault Testing of FPGAs," Journal of Electronic Testing, Theory, and Application, 2004.

[13] A. Krasniewski, M. Nowicka, "Application-Dependent Testability of FPGA-Based Circuits Designed Using Functional Decomposition," Proc. IFIP Workshop on Logic and Architecture Synthesis, pp. 167-175, 1996.

[14] A. Krasniewski, "Design for Application-Dependent Testability of FPGAs," Proc. Int'l Workshop on Logic and Architecture Synthesis, pp. 245-254, 1997.

[15] W. Quddus, A. Jas, N. A. Touba, "Configuration Self-Test in FPGA-Based Reconfigurable Systems," Proc. ISCAS'99, pp. 97-100, 1999.

[16] __, *The Programmable Logic Data Book 2003*, Xilinx Inc., 2003.

[17] M. L. Bushnell, V. D. Agrawal, *Essentials of Electrnonic Testing for Digital, Memory and Mixed-Signal VLSI Circuits*, Boston: Springer, 2006

[18] S. Vemula, C. Stroud, "Built-In Self-Test for I/O Buffers in FPGAs", Proc. NATW'05, 2005

[19] S. Vemula, C. Stroud, "Built-In Self-Test for Programmable I/O Buffers in FPGAs and SoCs", Proc. IEEE Southeastern Symp. on System Theory, pp. 534-538, 2006

[20] W. J. Huang, E. J. McCluskey, "Column-Based Precompiled Configuration Techniques for FPGA fault tolerance," Proc. IEEE Symp. on Field-Programmable Custom Computing Machines, 2001