BIST-Based Delay Path Testing in FPGA Architectures

Ian G. Harris, Premachandran R. Menon, and Russell Tessier
Department of Electrical and Computer Engineering
University of Massachusetts at Amherst
E-mail: harris@ecs.umass.edu, menon@ecs.umass.edu, tessier@ecs.umass.edu
Phone: 413-545-1594, 413-545-2515, 413-545-0160
Fax: 413-545-1993

Abstract—
The widespread use of field programmable gate arrays (FPGAs) as components in high-performance systems has increased the significance of path delay faults in FPGAs. We present a technique for FPGA path delay fault detection which integrates test insertion with the FPGA placement and routing stages to accomplish testing with low test application time. An accurate static timing analyzer is used to identify critical paths and built-in self-test (BIST) hardware is inserted using a placement and routing tool. Initial experimental results show that testing is accomplished with low test application time for several benchmark designs.

I. Introduction

Field programmable gate array (FPGA) technology has drastically reduced the cost of hardware manufacture, making hardware implementation economically feasible for applications which were previously restricted to software. As the use of FPGAs in commercial products becomes more commonplace, the significance of reliability and test has a greater financial impact. The use of FPGA technology in high-performance systems has increased the significance of FPGA path delay faults. As FPGA technology moves to cluster-based architectures [3], the major impact of delay faults on interconnect paths becomes more acute.

We present a built-in self-test (BIST) approach for the testing of FPGA path delay faults. In order to be able to guarantee correct operation at the highest possible speed, it is essential to verify that no path delay exceeds the clock period. FPGA BIST techniques do not suffer from the overhead restrictions of traditional BIST because the FPGA is reconfigured after testing to remove all BIST logic. Given the millions of interconnect transistors in an FPGA it would be impossible to test the path delay characteristics of all feasible interconnect paths. Most FPGA consumers load a specific design into many FPGAs. Therefore it is only necessary to test these specific design paths. Examples include FPGAs used in network hardware, data processing equipment, and military systems. Although devices have the capability to be reconfigured in the field, most FPGAs maintain the same design throughout their usable life. As a result initial testing for a specific design at manufacture time is of primary verification concern.

Our technique is application specific, testing only those paths which are critical for a particular application. A static timing analyzer is used to identify critical paths for testing and the paths are scheduled into test sessions each of which tests a set of paths in parallel. Since FPGA reconfiguration can require a great deal of time, the chief goal is to minimize the test application time by minimizing the number of test sessions. The main restriction on the insertion of BIST logic is that the delay characteristics of the paths under test must not be altered. This is accomplished by inserting BIST logic using only components of the FPGA which are not driven by any path under test. We have modified a placement and routing tool to consider this restriction during BIST insertion.

The remainder of this paper is organized as follows: Section II summarizes previous work. Section III provides a high level view of the system which we have implemented to insert BIST logic into an FPGA design. Section IV briefly reviews the assumed FPGA structure and the basic concepts of delay testing. Section V describes how BIST structures are inserted into the design to test each path. Section VI describes the process of test session definition. Sections VII and VIII present experimental results and conclusions respectively.

II. Previous Work

Several methods for non-delay testing FPGAs have been published in the literature, e.g., [2], [9], [8], [10], [11], [12], [16], [17], [19]. Most of the methods utilize built-in self-test (BIST) by configuring a pattern generator and a signature analyzer from unused parts of the FPGA. These methods have been applied to testing for logic blocks faults [10], [16], interconnect faults [15], [17], [8], [9] and bridging faults [19].

FPGA BIST techniques have also been applied to delay testing by using test pattern generators which generate two pattern tests [6], [14]. Krasniewski [11], [12], has proposed an approach which configures unused CLBs in the FPGA as linear feedback shift registers (LFSRs) for pattern generation and signature analysis. However, the delay fault coverage obtained using random patterns has been found to be low [12]. Two methods of improving fault coverage by modifying the the functions implemented by individual configurable logic blocks (CLBs) have been proposed. Since the delay through a CLB implemented by a look-up table (LUT) is independent of the function realized by it, this transformation will not affect any path delay.

In the first method [11], the LUT of every module is reprogrammed to implement the parity (XOR) function of its inputs. Although this transformation increases the probability of detection of delay faults, the signal transitions along the tested path may not be the same as those in the original circuit. Therefore, the test results may not truly indicate whether or not the original circuit would operate correctly at the rated clock speed.

The second method [12] attempts to correct this by reprogramming each LUT so that the transitions on each input/output pair are the same as those in the original LUT and the total number of output transitions is maximized. This method has been shown to produce higher fault coverage with reduced test sequence length. Since the functions implemented by the various CLBs have been changed, the signal transitions along various paths may still differ from those in the original circuit.

III. BIST Insertion System

The overall structure of the BIST insertion system is shown in Figure 1. A key part of our approach is the direct implementation of path delay testing circuitry in the FPGA device. To prove the delay testing concept we have used the Versatile Place and Route (VPR) FPGA tool set from the University of Toronto [4] to implement the circuitry. The tool suite includes timing driven packing, placement, and routing functions. The VPR tool targets a generic cluster-based architectural model rather than a specific industrial FPGA architecture.

The original logic design is synthesized using the VPR tool. A static timing analyzer which is built into the VPR tool [13] is used to identify critical paths for testing. Test sessions are defined by grouping together the critical paths which must be tested. The VPR tool is used again to place and route the BIST logic needed for each test session. The final result is a set of FPGA BIST designs, one for each test session. By programming the FPGA with each BIST design, all critical path timing faults are detected.

IV. Preliminaries

We assume an island-style FPGA architecture [5] which is composed of an array of identical tiles. Each tile is composed of a *cluster* [3] and surrounding interconnect as shown in Figure 2. 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. PIPs are shown as small squares at the intersections of line segments in Figure 2. Each cluster in a tile is a CLB whose functionality can be programmed. A CLB is composed of a set of multiplexers, flip-flops, and LUTs as shown in Figure 3. Each LUT is a 4-bit addressable RAM which can be programmed to implement any 4 input logic function.

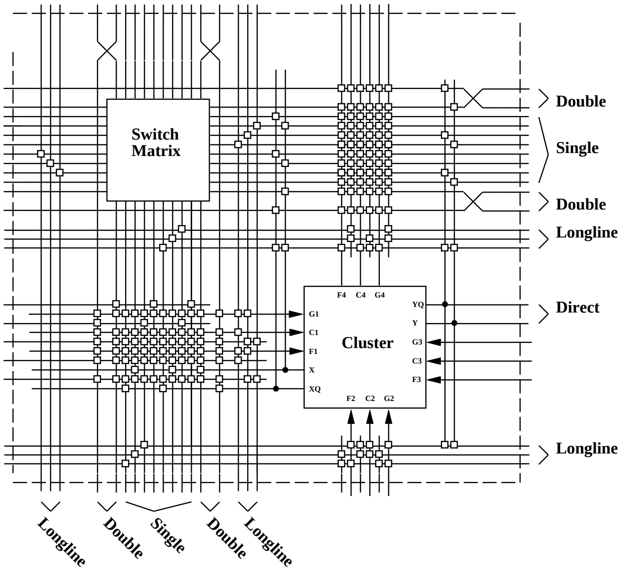


Fig. 2. Island-Style FPGA Tile

Before presenting our test method, we first review some terminology in delay testing, as applied to circuits consisting of simple gates (AND, NAND, OR, NOR and NOT). We shall use the same terminolgy for FPGAs, replacing gates by LUTs.

A *path* π is a sequence of gates and lines,

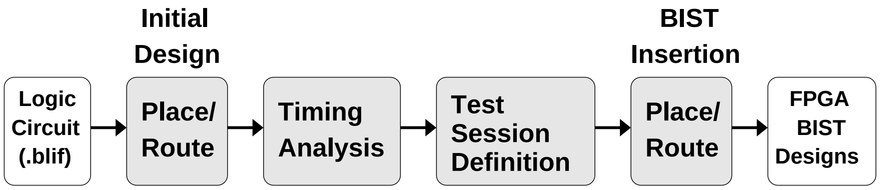


Fig. 1. FPGA BIST Insertion System

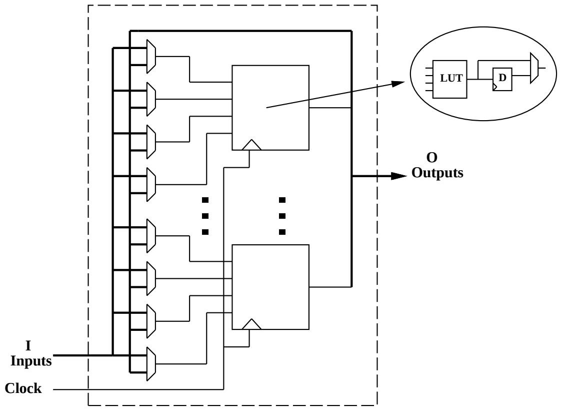


Fig. 3. A Configurable Logic Block

$g_0, l_0, g_1, l_1, ..., l_{n-1}, g_n$, where g_i is a gate, and l_i is a line that connects the output of g_i to an input of g_{i+1}. g_0 and g_n are the *source* and *destination* of π, respectively. We shall represent such a path by a sequence of gates alone, i.e., $\pi = g_0\, g_1 \ldots g_n$, since it is sufficient to uniquely identify the path. Every gate input on a path π is an *on-path input* of π. All other inputs to gates on π are called its *side inputs.*

A path π has a *rising (falling) transition fault* if, in the presence the fault, the delay of the path is greater than the clock period, for the particular direction of transition at the source of the path. An input vector v is said to *sensitize* a path if the value at the destination depends on the value at the source. A test for the rising(falling) transition fault on a path π consists of a vector-pair $< v_1, v_2 >$, where v_1 and v_2 produce $0(1)$ and $1(0)$, respectively, at the source of π, and v_2 sensitizes the path.

When path delay faults are caused by variations in the manufacturing process, we cannot assume that only the tested path is faulty. So, a frequent requirement in delay testing is that the fault under test must be detected independent of delays in the rest of the circuit. Tests with this property are called *robust tests.*

A test $< v_1, v_2 >$ is applied as follows: After v_1 is applied, sufficient time is allowed for the circuit to stabilize before v_2 is applied. This guarantees the initial value at the destination is that produced by the value at the source, independent of any delay faults that may be present. The value at the destination is observed (latched) exactly after the rated clock period. If the fault is present, the value at the destination will be the same as the value under v_1.

V. Circuit reconfiguration for testing

The paths to be tested are first identified by computing paths delays of the customized FPGA, using values available from FPGA specifications and routing information. Since the rise and fall delays are different, each selected path must be tested with the signal transitions that can be produced along the path under test during actual operation. However, the actual signal transitions will depend on the signal values that can be produced at the inputs of each CLB. Determining all combinations of signal transitions that may be produced along the path is therefore not feasible except in small circuits. Our goal is to test it for *all* transitions that can be produced along the path, based on the functions implemented by the LUTs on the tested path after customization. Our approach will determine whether the worst case delay of the path exceeds the test clock period.

We assume that the path to be tested starts at a flip-flop output, and ends at a flip-flop input. We first analyze each LUT along the path to determine whether it is positive unate, negative unate, or binate with respect to its on-path input. If it is positive unate, the LUT is changed to make the output equal to the on-path input, independent of the values of the remaining inputs. Similarly, each negative unate LUT is changed to make its outputs the complement of its on-path input. Every binate LUT is changed so as to produce the exclusive-OR of the on-path input and one of its other inputs, which we shall refer to as its *controlling side-input.* Since the delay through an LUT is independent of the function implemented by it, the above modifications will not affect the delay of the path being tested [11]. The transitions produced along the path will be the same as in the actual circuit if its side inputs were controllable.

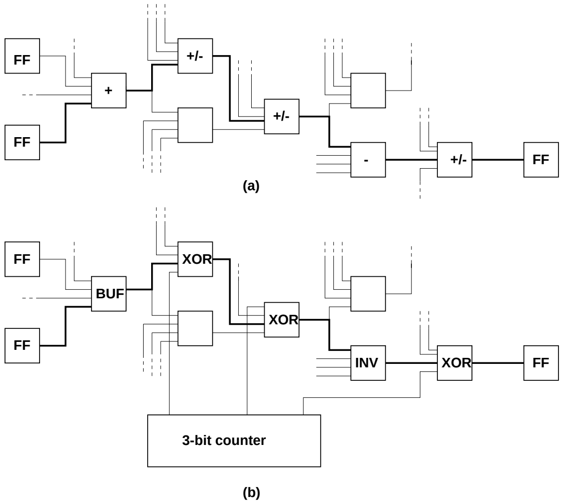


Fig. 4. Reprogramming LUTs for testing

The test circuitry is configured using CLBs that do not affect the delay of the path under test. For a path containing n binate LUTs, the test circuitry consists of an n-bit counter and control circuitry to produce transitions on the source flip-flop, and observe the destination flip-flop. The counter may be replaced by any component which produces all combinations of n bits in an arbitrary order. The outputs of the counter are connected to the controlling side-inputs of the binate LUTs. To prevent any change in the delay of the tested path, no part of the FPGA in the transitive fanout of LUTs on the tested path, up to the first flip-flop should be used for testing. This guarantees that the load of the tested path remains unchanged.

Fig. 4(a) shows the original circuit, and the path to be tested in thick lines. Positive unate, negative unate and binate functions are labeled +, - and +/-, respectively. The modified circuit is shown in Fig. 4(b). Test application consists of producing a transition at the source flip-flop and observing whether the destination changes within a clock period. After each test application the counter is incremented to apply a new pattern of signals to the XOR. Thus, 2^n tests are applied for each direction of transition at the source.

In the modified circuit any signal transition at the output of the source flip-flop of the tested path will propagate along the path and produce a transition at the destination flip-flop. If the delay of the path is less than the clock period, the new value will be latched into the destination flip-flop.

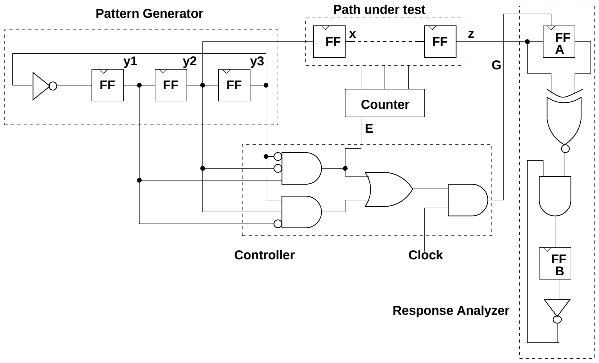
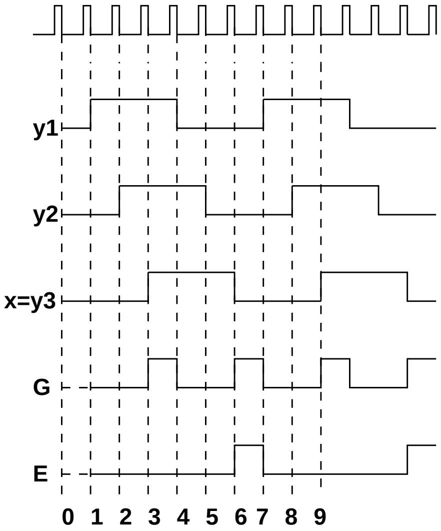


Fig. 5. Test configuration

Figure 5 shows the proposed test configuration for applying delay tests to a path in an FPGA. All the flip-flops shown are edge-trigerred, and clocked by a common clock. The pattern generator applies a repeated sequence of three 0s followed by three 1s to the flip-flop at the source of the path, as shown in Fig 6. At the clock pulse following every transition at the source of the path, labeled x in Fig. 5, the final value at the destination is latched into the destination flip-flop. Thus, the new and old values at the destination are available at the input and output, respectively, of FF-A. If they are equal, FF-B is set to 1, and will remain at that value until the end of the test session. Signal E which enables the counter to be incremented every six clock cycles, after both rising and falling transitions have been produced at x. Thus, the path is tested for all allowed combinations of inversion in all LUTs in it. A test session ends when the counter returns to the initial state.

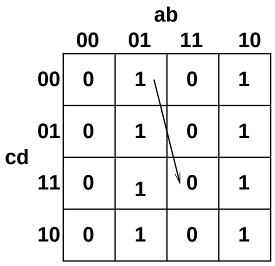
In the test procedure described above, the only signal transition applied during a test is at the source of the path under test. The controlling side-inputs of all LUTs are constant during the period when the transition propagates through the path. However, a side input of an LUT may change if it is from a fanout from the path under test. Since each LUT implements the XOR of its on-path input and controlling side-input, its output will be unaffected by any change in other side-inputs. This is illustrated in Fig. 7, where a tnd b are its on-path input and controlling side-input, respectively. Thus, delays along paths other than the one under test cannot affect the outcome of the test, and the test is robust.

The proposed method can be used to test a number of paths simultaneously if they are disjoint, except possibly a common source. The pattern generator and counter can be shared by all the tested paths. The number of bits in the counter must be the largest number of binate LUTs in any set of paths. Each path must have its own result analyzer, and their FF-Bs can be connected to form a scan chain. The failing paths can be identified by shifting out the contents of the



Time	Action
0-3	x stable at 0
3	x changes to 1; destination FF and FF-A will be at value produced by x=0
4	value produced by x=1 captured in destination FF
5	FF-B set if FF-A and destination FF have same value
6	x changes from 1 to 0 to test falling transition (x steady at 1 from 3 to 6)
7	final value of 1->0 transition captured in destination FF counter incremented

Fig. 6. Timing diagram



cd \ ab	00	01	11	10
00	0	1	0	1
01	0	1	0	1
11	0	1	0	1
10	0	1	0	1

Fig. 7. LUT implementing XOR

scan chain at the end of a test session. By restricting the placement of BIST logic to avoid all paths under test and their fanout, we ensure that the delays of the paths under test are not modified by BIST insertion.

It is highly unlikely that a delay fault in the BIST logic will impact the results of testing. The BIST logic required for the approach described here is simple and compact as we demonstrate with our experimental results. By choosing an appropriate design for the counter shown in Figures 4(b) and 5, all delay paths in our BIST implementation can be limited to contain a single LUT. Because the delay paths in the BIST logic are short, it is unlikely that a path delay fault in the BIST logic will impact test results.

The test application time with our method depends on the maximum number of LUTs along a path and the number of test sessions, but not on the circuit size or the number of paths. If the longest path has n LUTs and the number of sessions required is S, the test application time will be $6 \cdot S \cdot 2 * *n$ clock cycles. The method can therefore be expected to be applicable to relatively large circuits.

VI. Test Session Definition

An important goal of FPGA test insertion is to minimize test application time by reducing the number of test sessions. Ideally all delay paths would be tested in a single session, but this is often impossible because some pairs of delay paths cannot be tested in the same session. There are two conditions under which a pair of paths cannot be tested in the same configuration:

- **Two paths share a LUT** - If two paths share a logic component, then it is possible for faults along these two paths to interfere with each other, and possibly mask each other.
- **One path drives an off-path input of the other path** - The problem in this situation is that the off-path input possibilities of one of the paths cannot be fully explored. If path 1 drives an off-path input of path 2, then that off-path input cannot be driven by a counter. If a counter does not drive the off-path input, then we cannot guarantee that all off-path input combinations are explored during testing.

By identifying all pairwise scheduling conflicts between paths, we create a *conflict graph* in which each node represents a path to test, and each edge represents the existence of a conflict between two paths. The problem of defining a minimum set of test sessions is equivalent to solving the Graph K-Colorability problem [7] which is NP-complete for $K \geq 3$. We solve this problem using an iterative, greedy heuristic which adds paths to a test session by selecting the one with the least number of conflicts with other paths not currently in the test session. The use of the heuristic enables the problem to be solved tractably for a very large number of paths.

VII. Experimental Results

A key part of our approach is the direct implementation of path delay testing circuitry in the FPGA device. To prove the delay testing concept we have used the Versatile Place and Route (VPR) FPGA tool set from the University of Toronto [4] to implement the circuitry. The tool suite includes timing driven packing, placement, and routing functions.

Eight benchmarks from the MCNC benchmark suite [18] were targeted to FPGA devices with the same architecture and physical characteristics as 1.8V Xilinx Virtex devices [1]. Initially, circuits were mapped to logic blocks and placed and routed. Then static analysis along all circuit paths was performed to locate the longest paths in the circuit. Subsequently, the test circuitry shown earlier was synthesized from RTL to FPGA logic using the Synopsys FPGA compiler. This logic was physically integrated into the FPGA device, placed and routed. The results in Table I show that the test circuitry for each design takes up a small portion of overall space and is easily implemented on any FPGA architecture. Our experiments show that the circuits for path delay testing were able to route successfully using VPR.

The first four columns of Table I show information about the FPGA layout of each design excluding test logic. The *Virtex* column shows the name of the Xilinx Virtex part to which each design was mapped, and the *size* column shows the dimensions of that part in terms of tiles. The *CLB/wire* column shows both the number of CLBs and the number of wires used by the design in the FPGA. The first two *BIST Data* columns show the number of LUTs, flip-flops, CLBs, and wires used by the BIST logic in each test session. The *sess.* column shows the number of test sessions required to delay test all critical paths of each design. The greedy heuristic used to identify test sessions results in the minimum number of test sessions in all cases.

VIII. Conclusion

We have presented a BIST method for testing selected sets of paths in FPGAs. The paths are selected based on computed path delays after the array has been programmed, placed and routed. The tests identify paths whose delay exceed the clock period for any input/output transition that can be produced along the tested path.

The method transforms the LUTs along the tested path to inverters, buffers or XORs, depending on the unateness of the original functions, allowing the path to be tested by simply applying transitions at the source of the path. A counter is used to apply all possible input combinations to the side inputs of the path. A number of non-interacting paths are tested simultaneously, sharing the pattern generator and control logic, and a scan chain is used to make test results of individual paths observable. Thus, the method not only detects faults on all tested paths but also identifies faulty paths. This information can be used during timing verification to reroute failing paths or modify the design.

Our experimental results have shown the feasibilty of the proposed method for relatively small circuits. As mentioned earlier, the test application time depends only on the maximum path length and the number of test sessions required to cover all selected paths. Although the CPU time for determining test sessions may increase with circuit size, the test application time is not likely to increase significantly.

References

[1] *Virtex Data Sheet.* Xilinx Corporation, 2000.

[2] 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.

[3] V. Betz and J. Rose. Cluster-Based Logic Blocks for FPGAs: Area-Efficiency vs. Input Sharing and Size. In *IEEE CICC*, pages 551–554, 1997.

[4] V. Betz and J. Rose. VPR: A New Packing, Placement and Routing Tool for FPGA Research. *Field-Programmable Logic and Applications*, pages 213–222, September 1997.

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

[6] C-A. Chen and S.K. Gupta. Design of efficient BIST test pattern generators for delay testing. *IEEE Transactions on Computer-Aided Design*, pages 1569–1575, 1996.

[7] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman and Company, 1979.

[8] I. G. Harris and R. Tessier. Diagnosis of Interconnect Faults in Cluster-Based FPGA Architectures. In *International Conference on Computer-Aided Design*, November 2000.

[9] I. G. Harris and R. Tessier. Interconnect Testing

Functional Data				BIST Data		
design	Virtex	size	CLB/wire	LUT/flop	CLB/wire	sess.
alu4	XCV50E	20x30	391/1018	77/39	21/66	2
apex2	XCV50E	20x30	491/1438	77/39	21/66	2
diffeq	XCV50E	20x30	379/1180	52/27	14/46	9
clma	XCV600E	48x72	2133/6134	94/47	25/78	4
frisc	XCV200E	28x42	894/2280	126/67	33/102	10
elliptic	XCV200E	28x42	906/2450	126/67	33/102	10
seq	XCV50E	20x30	457/1320	52/27	14/46	1
tseng	XCV50E	20x30	166/828	26/15	7/25	10

TABLE I

BIST Results for Several Benchmarks

in Cluster-Based FPGA Architectures. In *Design Automation Conference*, June 2000.

[10] W. K. Huang, F. J. Meyer, X.-T. Chen, and F. Lombardi. Testing Configurable LUT-Based FPGAs. *IEEE Transactions on Very Large Scale Integration Systems*, 6(2):276–283, June 1998.

[11] A. Krasniewski. Application-dependent testing of FPGA delay faults. In *EUROMICRO'99*, pages 260–267, 1999.

[12] A. Krasniewski. Enhancing detection of delay faults in FPGA-based circuits by transformations of LUT functions. In *IFAC Workshop on Programmable Devices and Systems – PDS2000*, pages 127–132, February 2000.

[13] A. Marquardt, V. Betz, and J. Rose. Timing-Driven Placement for FPGAs. In *International Symposium on Field Programmable Gate Arrays*, 2000.

[14] S. Pilarski and A. Pierzynska. BIST and delay fault detection. In *International Test Conference*, pages 236–242, 1993.

[15] 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.

[16] C. Stroud, E. Lee, and M. Abramovici. BIST-Based Diagnostics of FPGA Logic Blocks. In *International Test Conference*, pages 539–547, November 1997.

[17] 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.

[18] Steven Yang. Logic Synthesis and Optimization Benchmarks. *Microelectronics Centre of North Carolina Tech. Report*, 1991.

[19] 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.