Design-Specific Path Delay Testing in Lookup Table-based FPGAs

Premachandran R. Menon, Weifeng Xu, and Russell Tessier

***Abstract*—**

Due to the increased use of field programmable gate arrays (FPGAs) in production circuits with high reliability requirements, the design-specific testing of FPGAs has become an important topic for research. Path delay testing of FPGAs is especially important since path delay faults can render an otherwise fault-free FPGA unusable for a given design layout. This paper presents a new approach for FPGA path delay testing which partitions target paths into subsets that are tested in the same test configuration. Each path is tested for all combinations of signal inversions along the path length. Each configuration consists of a sequence generator, response analyzer and circuitry for controlling inversions along tested paths, all of which are formed from FPGA resources not currently under test. Two algorithms are presented for target path partitioning to determine the number of required test configurations. Test circuitry associated with these methods is also described. The results of applying the methods indicate that our path delay testing approach requires seconds per design to cover all paths with delay within 10% of the critical path delay. The approach has been validated using Xilinx Virtex devices.

I. Introduction

Field programmable gate arrays are widely used, not just for rapid prototyping, but also for production circuits. The testing of FPGAs has therefore become an important topic of research. FPGA tests are of two types – *manufacturing tests* and *user tests*. The former, performed as part of the manufacturing process, test components and interconnections in the array for faults, such as stuck-ats, shorts and opens. Components may also be tested to determine their switching speeds. User tests are intended to detect FPGA faults that occur after a device is programmed for a specific application. The faults of interest in this type of testing are only those that can affect the operation of the specific circuit. These consist of stuck-at faults, shorts, opens, and faults that affect circuit timing, usually called delay faults. Faults to be tested by user tests depend not only on the logic implemented by the circuit, but also on its placement and routing in the FPGA.

This paper is concerned with testing paths in lookup-table (LUT) based FPGAs after they have been routed. While this may be regarded as user testing by the above definition, we are considering an environment in which a large number of manufactured FPGA devices implementing a specific design are to be tested to ensure correct operation at the specified clock speed. It is thus akin to manufacturing tests in that the time needed for testing is important. Ideally, we would like to verify that the actual delay of every path between flip-flops is less than the design clock period. Since the number of paths in most practical circuits is very large, testing must be limited to a smaller set of paths. Testing a set of paths whose computed delay is within a small percentage of the clock period may be sufficient in most cases. Thus, our goal is to determine by testing whether the delay along any of the paths in the set exceeds the clock period. Our approach is general enough to target a range of contemporary LUT-based FPGAs, including recent commercial offerings from Xilinx and Altera.

P. Menon, W. Xu and R. Tessier are with the Department of Electrical and Computer Engineering, University of Massachusetts, Amherst, MA, 01003 USA

II. Previous Work

Several methods for testing field programmable gate arrays for faults other than delay faults have been published in the literature, e.g., [1], [2], [3], [4], [5], [6], [7], [8], [9]. Most of these methods utilize built-in self-test (BIST) by configuring a pattern generator and a signature analyzer from unused FPGA resources. These methods have been applied to test for logic block faults [5], [6], interconnect faults [4], [7], [9] and bridging faults [8]. BIST techniques have also been applied to delay testing, using test pattern generators which generate two-pattern tests [10], [11]. However, the fault coverage obtained by applying linear feedback shift register (LFSR) generated tests was found to be low.

Krasniewski [12], [13], [14], [15] has proposed a number of techniques for improving the fault coverage obtained by pseudo-random testing. These methods are based on reprogramming logic blocks on the paths under test so as to facilitate testing. This re-programming will not affect path delays since the delay through a logic block implemented by a LUT is independent of the function implemented by it. In [12], every logic block is re-programmed to implement the parity 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. The test results may not truly indicate whether the original circuit would operate correctly at the rated clock speed, since path delays may depend on the direction of signal transitions along it. This deficiency is corrected in [13] by re-programming each logic block so that all input/output signal pairs in the original LUT are also contained in the modified LUT, and the number of output transitions is maximized. This, however, led to some conditions where signal propagation was blocked along tested paths. An improved method proposed in [14] facilitates propagation of fault effects by reducing such blocking conditions. While these methods have been shown to produce very high fault coverage with reduced test sequence length, they cannot guarantee that all delay faults will be detected.

A comparison-based delay test method is proposed in [16]. A number of identical paths are constructed in the FPGA under test and every LUT on these paths is programmed to propagate an input value to its output. The same transition is applied simultaneously to the inputs of all these paths. A fault is detected when the difference between the arrival times at the destinations of the first and last signals exceeds a specified threshold. The accuracy of the method depends on the variation in delay of different segments, the length of the paths and the allowed difference between the fastest and slowest paths. Moreover, the method does not test for delay faults in paths in the customized circuit, but only the path delay along specific paths. A similar technique for FPGA interconnect delays is proposed in [17]. Using an iterative logic array model, it tests a number of similar sections of interconnects simultaneously, and also provides information to locate faulty sections.

Tahoori and Mitra [18] have proposed a method of testing all paths in a combinational network for delay faults with only two tests. By changing all LUTs to implement the AND of their inputs, all paths are simultaneously tested for slow-to-rise faults by applying 0->1 transitions at all inputs. Slow-to-fall faults are similarly tested by changing LUT functions to ORs and applying 1->0 transitions at all inputs. A 4-phase test method based on the same principle has also been proposed for sequential circuits. While this approach is very efficient in the number of tests needed, all interconnects undergo signal transitions in the same direction during each test. Since the delay of an interconnect depends on the direction of signal transition, this method cannot test the paths for the combinations of transitions that occur during normal operation.

III. Basic Approach

Before presenting the proposed test method, we define several terms. Our method is applicable to FPGAs in which the basic logic elements are implemented by look-up tables.

A. Definitions

A *path* consists of a sequence $\{l_0, c_0, l_1, ..., c_{n-1}, l_n\}$ of LUTs l_i and connections c_i, such that c_{i-1} is an input of l_i, $1 \leq i < n$. Line c_{i-1} is called the *on-path input* of LUT l_i. All other inputs of l_i are called its *side inputs*. The paths to be tested start and end at flip-flops, which are called the *source* and *destination flip-flops* (or simply source and destination), respectively.

The goal of this work is to test a set of paths, called *target paths*, to determine whether the maximum delay along any of them exceeds the clock period of the circuit. These paths are selected based on static timing analysis using nominal delay values and actual routing information. Circuitry for applying test patterns and observing results is configured using parts of the FPGA that are not under test.

B. Introduction to Approach

The delay of a path segment usually depends on the direction of signal transition in it. The direction of the signal transition in any segment is determined by that of the transition at the source and the inversions along the partial path leading to the particular segment. A test to determine whether the maximum delay along a path is greater than the clock period must propagate a transition along the path and produce a combination of side-input values that maximizes the path delay. This approach is not usually feasible because of the difficulty of determining the inversions that maximize the path delay and the necessary primary input values to produce them. Instead, we propose to test each target path for all combinations of inversions along it, guaranteeing that the worst case will also be included. Although the number of combinations is exponential in the number of LUTs along the path, the method is feasible because application of each test requires only a few cycles of the rated clock. However, the results may be pessimistic in that a path that fails a test may operate correctly in the actual circuit, because the combination of inversions in the failing test may not occur during normal operation.

We shall first explain our method of testing a single path in a circuit and describe a test circuit for implementing it. Application of this method to test a number of paths simultaneously is discussed in the next section.

Our approach, first suggested in a recent paper [19], reprograms the FPGA to isolate each target path from the rest of the circuit and make inversions along the path controllable by an on-chip test controller. Every LUT along the path is re-programmed based on its original function. If it is positive unate in the on-path input, the LUT output is made equal to the on-path input independent of its side inputs. Similarly, negative unate functions are replaced by inverters. If the original function is binate in the on-path input, the LUT is re-programmed to implement the exclusive-OR (XOR) of the on-path input and one of its side-inputs, which we shall call its *controlling side-input*. As mentioned earlier, this change of functionality does not affect the delay of the path under test because the delay through an LUT is unaffected by the function implemented. Inversions along the path are controlled by the signal values on the controlling side-inputs. For each combination of values on the controlling side inputs, we apply a signal transition at the source of the path and observe the signal value at the destination after one clock period. The absence of a signal transition will indicate that the delay along the tested path exceeds the clock period for the particular combination of inversions.

The basic method described above can be implemented by the circuitry shown in Fig. 1, consisting of a sequence generator, a response analyzer and a counter, that generates all combinations of values in some arbitrary order. A linear feedback shift register modified to include the all-0's output [20], [21] may be used as the counter. The controlling side inputs are connected to the counter. The controller and the circuitry for applying tests and observing results are also formed during configuration in parts of the FPGA that do not affect the behavior of the path(s) under test.

The sequence generator produces a sequence of alternating 0's and 1's, with period equal to $6T$, where T is the operational clock period. The response analyzer checks for an output transition for every test, and sets an error flip-flop if no transition is observed at the end of a test. The flip-flop is reset only at the beginning of the test session, and will indicate an error if and only if no transition is produced in some test. The counter has as many bits as the number of binate LUTs along the tested path.

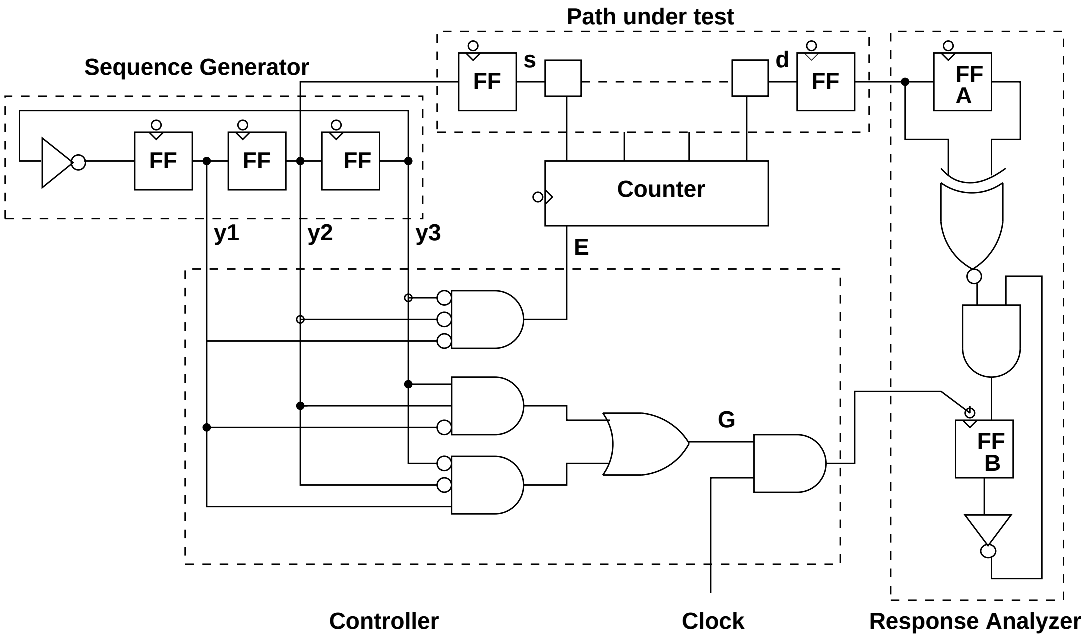
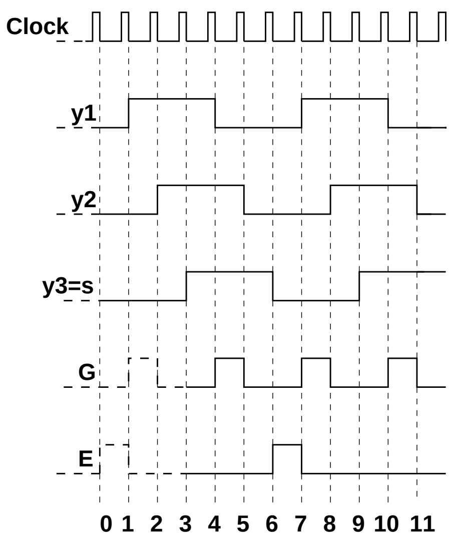


Fig. 1. Testing a single path for a negative edge-triggered design. For a positive edge-triggered design, all negative edge-triggered components are replaced by positive edge-triggered counterparts.



Time	Action
0−3	s = 0
3	Destination FF(d) stable at value produced by s=0; s = 0−>1
4	Final value of 0−>1 transition captured in destination FF (d)
5	FF−B set if FF−A and destination have same value
6	s = 1−> 0 to test falling transition; counter enabled
7	Final value of 1−>0 transition captured in destination FF (d); counter incremented
8	FF−B set if FF−A and destination have same value
9	s = 0−>1 to test rising transition with next set of inverstions

Fig. 2. Timing diagram for clock with period T

The test for a path for each direction of signal transition consists of two parts, an initialization part and a propagation part, each of duration $3T$. A path is tested in time $6T$ by overlapping the initialization part of each test with the propagation part of the preceding test. In addition, the change of counter state for testing a path for a new combination of inversions is also done during the initialization phase of rising transition tests.

Figure 2 shows the timing of the signals during application of a test sequence. It can be seen from the figure that the source s of the test path toggles every three clock cycles. For correct operation, the input transition occurring at $3T$ must reach the destination within time T (i.e., before $3T + T$). On the following clock edge at $3T + T$, the result of the transition is clocked into the destination flip-flop at d. A change must be observed at the destination for every test, otherwise a flip-flop is set to indicate an error. In Figure 2, a test for the rising edge starts at time $3T$, with the s steady at 0 for the preceding three clock cycles. A test for the falling transition starts at $6T$, with the input steady at 1 for the preceding three clock cycles. Results are sampled at d at time $4T$ (for rising edge s transition) and $7T$ (for falling edge s transition), respectively. Thus, both rising and falling transitions are applied at the source for each combination of inversions in time $6T$.

As the falling transition is applied at $6T$, the enable input E of the counter is set to 1. This action starts a state (counter) change at $7T$ to test the path for the next combination of inversions. A counter change at this time point allows $2T$ of settling time before the following transition occurs at the source s. By ensuring that the counter reaches its final value within T and propagates to the path destination d within an additional T, d

is ensured to be stable before the following source transition. Thus, the destination will reach the correct stable value corresponding to the new combination of inversions if no path from the counter to the destination has a delay greater than $2T$. This delay explains the need for a $3T$ period between s transitions ($1T$ to perform the test, $1T$ for possible counter state changes, and $1T$ for subsequent propagation of the counter change to d).

IV. Test Strategy

The method described in the preceding section requires the test control circuitry to be re-configured for every path to be tested. The total time for testing a set of target paths in a circuit consists of the test application time and the re-configuration time. Our goal is to reduce both components of the total time for testing a specified set of paths. Since the time needed for configuring the test structure is usually larger than that for applying test patterns generated on-chip, we shall focus on reducing the number of test configurations needed by testing as many paths as possible in each configuration.

Two approaches to maximize the number of paths tested in a test configuration suggest themselves. First, we can try to select a set of target paths that can be tested simultaneously. This will also have the effect of reducing test application time. Secondly, we can try to select a set of simultaneously testable sets that can be tested in sequence with the same configuration. In this case, the number of simultaneously tested paths may have to be reduced so as to maximize the total number of paths tested with the configuration. These two approaches will be elaborated in the next two subsections, but first we define a few terms.

The simultaneous application of a single rising or falling transition at the sources of one or more paths and observing the response at their destinations is called a *test*. The set of tests for both rising and falling transitions for all combinations of inversions along each path is called a *test phase*, or simply, a *phase*. As mentioned earlier, a single path with k binate LUTs will have $2 \cdot 2^k$ tests in a test phase. The application of all test phases for all target paths in a configuration is called a *test session*.

A. Single Phase Method

This method, first presented in [19], attempts to maximize the number of simultaneously tested paths. A set of paths may be tested in parallel if it satisfies the following conditions:

1) No two paths in the set have a common destination.
2) No fanout from a path reaches another path in the set.

The above conditions guarantee that signals propagating along paths in the set do not interfere with one another. Moreover, if the same input is applied to all paths in the set, two or more paths with a common initial segment will not interact if they do not re-converge after fanout.

All LUTs on paths to be tested in a session are reprogrammed to implement inverters, direct connections or XORs as discussed in the preceding section. The LUTs with control inputs are levelized, and all control inputs at the same level are connected to the same counter output. The source flip-flops of all paths to be tested in the session are connected to the same sequence generator, but a separate transition detector is used for each path. The transition detectors of all paths are then ORed together to produce an error indication if any of the paths is faulty. Alternatively, a separate error flip-flop can be used for each tested path, connected to form a scan chain and scanned out to identify the faulty path(s).

B. Multi-phase Method

The single phase method described above requires that all paths tested in a session be disjoint. The number of test sessions needed for a large target set is therefore likely to be very large. The multi-phase method attempts to reduce the number of test sessions needed by relaxing the requirement that all paths tested in a session be disjoint. This, however, increases the test application time because non-disjoint target paths may interact and cannot be tested simultaneously.

Consider sets of target paths S_1, S_2, ..., S_p such that all paths in each set are disjoint except for common sources. Clearly, all paths in each set S_i can be tested simultaneously, as in the single phase method, if each set can be selected and logically isolated from all other paths. This allows the testing of the sets S_i in sequence, and is the basis of our multi-phase method. We also restrict the target paths for each session to simplify the control circuitry needed.

We assume that the LUTs in the FPGA are 4-input LUTs, but the method can be easily modified to allow a larger number of inputs. Since each LUT may need up to two control inputs, one for path selection and the other for inversion control, at most two target paths may pass through any LUT. Target paths satisfying the following conditions can be tested in a single session.

1) There is a path to each target path destination, called the *main path* to the destination.
2) Main paths may not intersect, but they may have a common initial section.
3) Additional paths to each destination, called its *side paths*, must meet only the main path and continue to the destination along the main path.
4) Main paths and side paths may not intersect any other path, except that two or more paths may have a common source.
5) No more than two target paths may pass through any LUT.
6) The number of target paths to all destinations must be the same.

The above conditions allow us to select one path to each output and test all of them in parallel. The first two conditions guarantee that the signal propagating along main paths to different destinations will not interact. The main paths can therefore be tested in parallel. The restriction that a side path can meet only the main path to the same destination (Condition 3) allows a simple mechanism for propagating a signal through the main path or one of its side paths. Together with Condition 4, it guarantees that a set of main paths or a set of side paths, one to each destination, can be tested in parallel. Condition 5 allows for two control signals to each LUT, one for controlling inversion, and the other for selecting the path for signal propagation. A single binary signal is sufficient for selecting one of the target paths that may pass through an LUT. The last condition is required to produce a signal change at every destination for every test, simplifying the error detection logic.

With the above restrictions, LUTs on target paths will have one or two target paths through them. These LUTs are called *1-path LUTs* and *2-path LUTs*, respectively. The inputs that are not on target paths will be called *free inputs*.

The following procedure selects a set of target paths satisfying the conditions for multi-phase testing by selecting appropriate target paths for each set S_i from the set of *all* target paths in the circuit. The union of these sets is the set of paths targeted in a test session. The procedure is then repeated for the remaining paths to obtain the target paths for subsequent test sessions until all paths are covered.

Procedure 1

1) Select a path that does not intersect any already selected path, as the main path to each destination.
2) For each main path, select a side path such that
   a) It meets the main path and shares the rest of the path with it.
   b) No other path meets the main path at the same LUT.
   c) It does not intersect any already selected target path (except for segments overlapping the main path).
3) Repeat Step 2 until no new side path can be found for any main path.
4) Find the number, n, of paths such that
   a) There are n target paths to each destination.
   b) The total number of paths is a maximum.
5) Select the main path and $n-1$ side paths to each destination as the target paths for the session.

Example 1: Figure 3 shows all the target paths in a circuit. The source and destination flip-flops are omitted for the sake of clarity. We start Procedure 1 by (arbitrarily) selecting $dAEJLy$ and $hCGKMz$ as the main paths to the destinations y and z. Adding paths $eAEJLy$, $cEJLy$ and $fBFJLy$ to the first path, and $jCGKMz$, $nDGKMz$ and $qHKMz$ to the second, we get the set of target paths shown in heavy lines. Since there are four paths to each destination, the eight target paths shown can be tested in a single four-phase session.

The procedure can be repeated with the remaining paths to select sets of target paths for subsequent sessions. One possible set of test sessions is given in the following table, where the path(s) in the first row of each sessions were those chosen as the main path(s).

	Destination: y	Destination: z
Session 1	$dAEJLy$	$hCGKMz$
	$eAEJLy$	$jCGKMz$
	$cEJLy$	$nDGKMz$
	$fBFJLy$	$qHKMz$
Session 2	$gBEJLy$	$gHKMz$
	$gFJLy$	$kDGKMz$
Session 3	$gBFJLy$	$mDGKMz$
Session 4	$hCFJLy$	
	$jCFJLy$	
	$kDGLy$	
Session 5	$nDGLy$	
Session 6	$mDGLy$	

The set of sessions may not be unique and depends on the choices made. Also note that not all sessions obtained are multi-phase sessions. Session 3, for example, became a single-phase session because no path qualified as a side path of $mDGKMz$, which was arbitrarily chosen as the main path. No paths could be concurrently tested with those in Sessions 4, 5, and 6 because all paths to z had already been targeted.

The sets of target paths obtained by Procedure 1 are such that each 2-path LUT has a main path and a side path through it. Thus, a single binary signal is sufficient to select the input through which the signal is to be propagated. Since the side path continues along the main path, selecting the appropriate input at the 2-path LUT where it meets the main path is sufficient for selecting the side path for testing. By using the same path selection signal, one side path to each destination can be selected simultaneously and tested in parallel.

The FPGA configuration for a test session is obtained by the following procedure:

Procedure 2

1) Configure a sequence generator and connect its output to the sources of all target paths of the session.
2) Configure a counter to control inversion parity, with the number of bits equal to the largest number of binate LUTs along any target path for the test session.
3) Configure a path selector to select the set of paths tested in each test phase, with the number of bits equal to the number of side paths to a destination.
4) Designate a free input of each LUT as its inversion control input p, and connect it to the counter output corresponding to its level.
5) Designate another free input of each 2-path LUT as its *selector* input s, and connect it to the path selector.
6) Modify the LUT of each 1-path LUT with on-path input a to implement $f = a \oplus p$, if the original function is binate in a; otherwise $f = a$ if it is positive or $\overline{a}$ if it is negative in a.
7) Modify the LUT of each 2-path LUT to implement $f =$

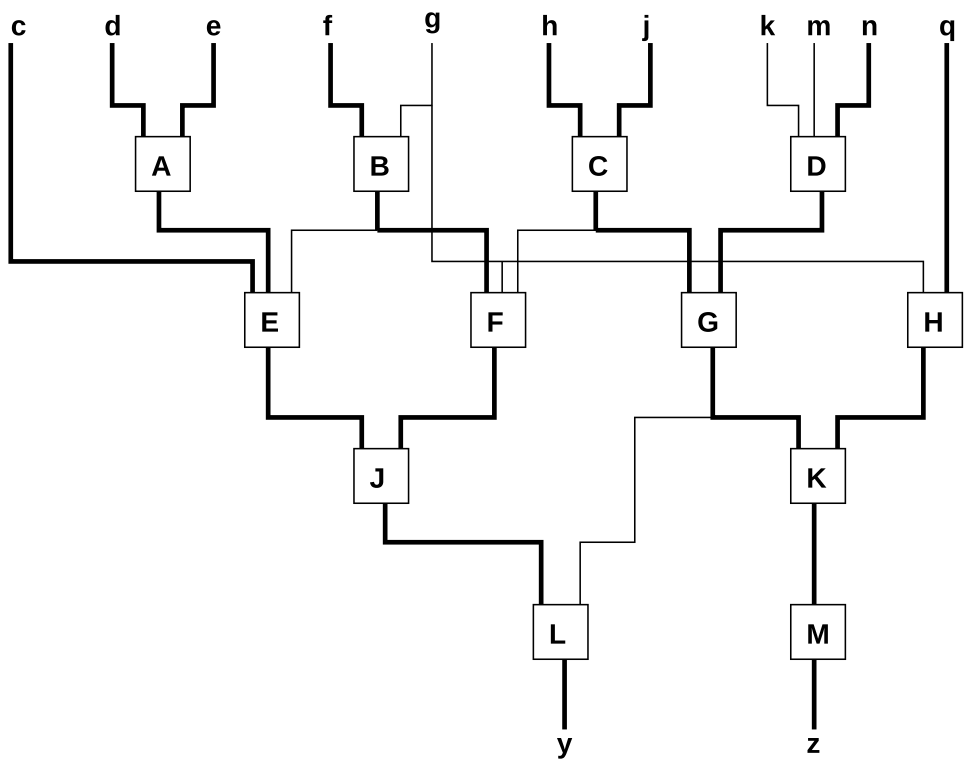


Fig. 3. Selected target paths

$\overline{s} \cdot (a \oplus p) + s \cdot (b \oplus p)$, where a and b are on the main path and a side path, respectively.

The above modification for 2-path LUTs assumes that they are binate in both on-path inputs. If the output of a 2-path LUT is unate in a or b or both, a slightly different function f is needed. For example, if the LUT output is binate in a and negative in b, the modified LUT must implement $f = \overline{s} \cdot (a \oplus p) + s \cdot \overline{b}$.

Example 2:

Figure 4 shows the test structure for the circuit of Fig. 3. Only target paths that were selected for the first test session are shown, and all LUT functions are assumed to be binate in their inputs. The test circuitry consists of a sequence generator that produces a sequence of alternating 1's and 0's, a four-bit counter for inversion control and a path selector. The path selector is a shift register that produces an output sequence, 000, 100, 010, 001 for the 4-phase test of the first session in our example.

It can be verified from the figure that the main paths are selected when all selector outputs are 0. When any output is 1, exactly one side path to each destination is selected. Input transitions are applied to all paths simultaneously, but propagate only up to the first 2-path LUT on all paths except the selected ones. Thus, only one path to each destination will have transitions along its entire length. Since these paths are disjoint, no interaction can occur among them.

C. Test Time for Single and Multi-Phase Methods

The total test time for all target paths using the single phase or multi-phase method can be computed as follows: Let n be the number of test sessions. Let p_i be the number of phases in the i^{th} session and k_i the largest number of LUTs with control inputs among the paths tested in the session. The i^{th} session will have $p_i \cdot 2 \cdot 2^{k_i}$ tests requiring $6 \cdot p_i \cdot 2^{k_i}$ clock cycles. If T_c is the reconfiguration time per test session, the total test time is is given by:

$$6 \cdot T \cdot \sum_{i=1}^{n} p_i \cdot 2^{k_i} + n \cdot T_c.$$

For the single phase method, $p_i = 1$ for all i, and the above formula reduces to:

$$6 \cdot n \cdot T \cdot \sum_{i=1}^{n} 2^{k_i} + n \cdot T_c = n \cdot (6 \cdot T \cdot \sum_{i=1}^{n} 2^{k_i} + T_c).$$

D. Limitations and Assumptions

Several assumptions were made in the development and evaluation of our approach:

- **Constant LUT delay** - As mentioned earlier, for this work we assume LUT delays are independent of the function they implement. Although this assumption has recently been drawn into question [22] for certain logic functions, it is accurate for most functions. Additionally, the fraction of the FPGA path delay within LUTs is usually a small percentage (less than 5%) of overall path delay.
- **LUT-only logic** - The current approach is only targeted to paths which connect LUT logic resources. Although testable paths including non-LUT logic, such as fixed adder carry chains, could be determined via the approach outlined in [23], the programming of off-path inputs to propagate signal transitions would require additional circuitry.
- **Effects of crosstalk** - Left unaddressed, crosstalk can have a significant impact on delay in FPGA circuits. To address this issue, commercial FPGA routers actively control the

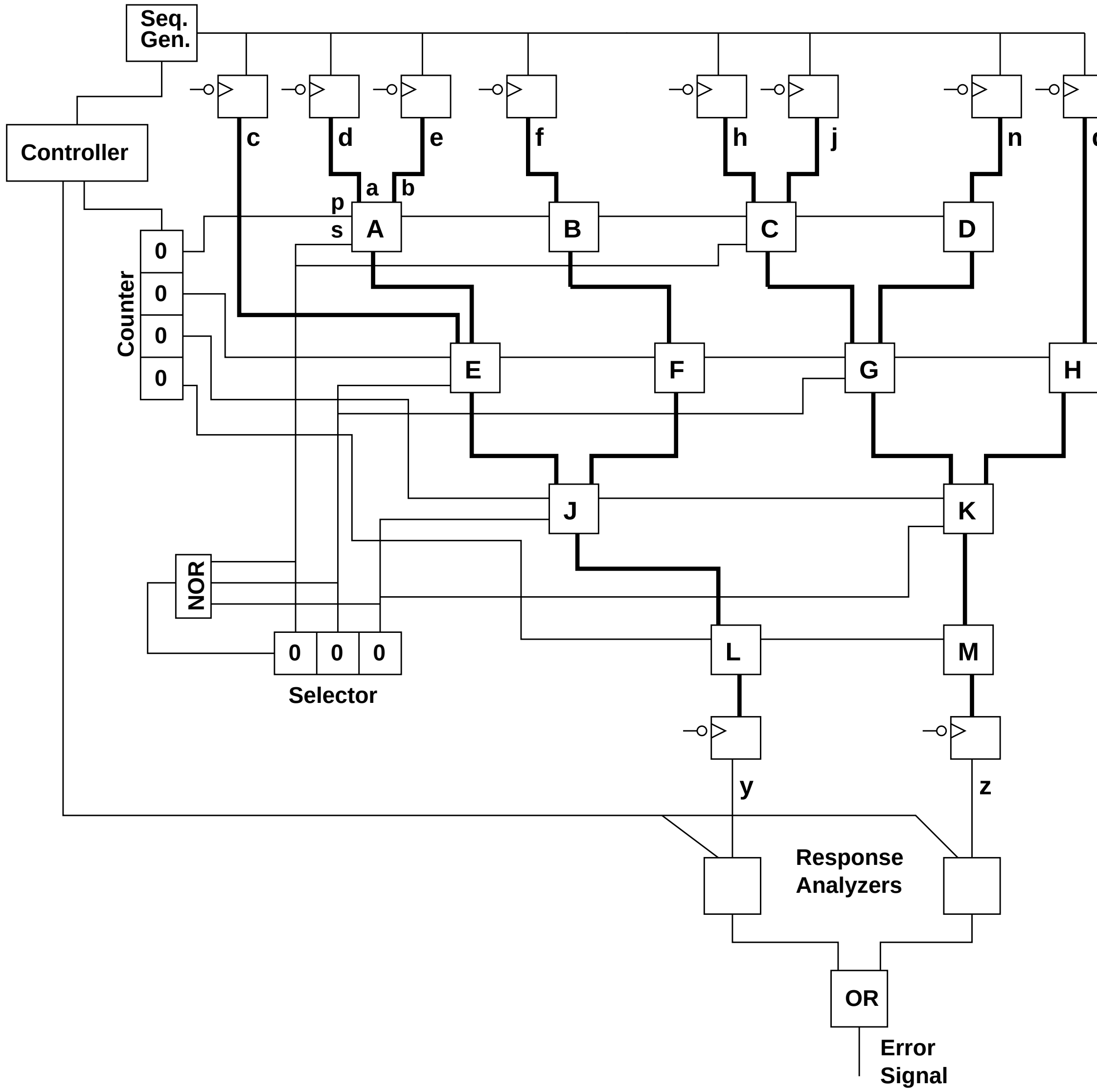


Fig. 4. Multi-phase test structure

assignment of design nets to wires to limit the effect of crosstalk. As a result, we neglect the possible delay effects of crosstalk in this work.

- **Isolation of test circuitry** - Since all test circuitry is implemented using wires and logic that are unused by the paths under test during a test session, this circuitry does not affect the size of circuit nor the number of paths that can be tested.
- **Delays in test circuitry** - It was mentioned in Section III-B that our design allows $2T$ for the circuit to settle after a change of counter state. This guarantees the validity of the tests even in the presence of delay faults in the test circuitry, provided that the above restriction is satisfied. The restriction can be relaxed by increasing the test cycle period from $6T$ to $8T$, thus allowing a settling time of $3T$. The test cycle period can be increased by adding one flip-flop to the sequence generator.

Currently, it is not possible to make incremental changes in test sessions in response to incremental design changes. The entire test procedure must be restarted following any design modifications.

V. Experimental Approach

To demonstrate the benefits and costs of our path delay testing approach, a fully integrated computer-aided design (CAD) system was developed. This system incorporates an implementation of the algorithms described in Section IV with commercial and academic FPGA synthesis and physical layout tools. The following specific steps are performed in order to determine the number of test sessions required and the paths assigned to each test session.

- The design under test is synthesized, technology mapped, placed, and routed using FPGA CAD software.
- A static analyzer is run on the placed and routed design to enumerate all design paths in terms of path delay. All paths which have delay within 10% of the critical path delay are selected for test. This path selection approach has been used in several previous studies [24] [25].
- The test algorithms outlined in Section IV are applied to the circuit to determine the test session and phase for each

path in the test set. This effectively assigns each tested path to a specific test session.

Each circuit was technology mapped to a target Xilinx Virtex FPGA using Xilinx XSE tools and placed and routed using an enhanced version [26] of the Versatile Placement and Routing (VPR) FPGA tool suite [27]. A static timing analyzer developed for VPR was then used to identify the paths with delay within 10% of the critical path delay. Existing commercial Xilinx XSE tools could have also been used for place and route and static analysis.

To perform the path delay tests in the FPGA hardware, one FPGA circuit is required per test session. This circuitry consists of the logic and routing resources for the paths under test and the test circuitry shown in Figs. 1 and 4. Specific steps performed to create each test session circuit include:

- Test circuitry for the test session is synthesized, technology mapped and clustered using Xilinx XSE and VPR tools. This circuitry is subsequently combined with the path wires and logic blocks under test by merging the netlists with a script.
- The combined test session circuitry is placed and routed using our modified VPR system. With this system, it is possible to constrain the tested paths to the routing wires, multiplexers, and internal logic block connections used in the original design routing so that the target path delays of the original design can be tested. Added test circuitry is implemented in logic that is not used by the paths under test.
- Final test session placement and routing information is input to Xilinx XSE tools to verify valid chip routing and to verify that all test circuitry operates at the target clock frequency. A configuration bitstream for the test session is then created.

The middle step of this 3-step process was performed with academic place and route tools due to the difficulty of reading and manipulating detailed routing information that is generated by commercial FPGA CAD tools. Although the Xilinx XSE tools provide a straightforward user interface to allow for the pre-defined placement of logic, it is difficult to determine the exact routing resources used by paths during the original design route and to constrain the route of these paths during the subsequent route of test session circuits. Commercial tools could be used in place of VPR for our approach if commercial FPGA CAD software provided straightforward interfaces to read and constrain path routing. Alternatively, our test session circuit creation methodology could easily be integrated into a commercial FPGA CAD flow by a tool vendor as a test generation flow option.

VI. Experimental Results

To validate our approach with the steps outlined in Section V, we evaluated a series of MCNC [28] and RAW [29] benchmark circuits targeted to Xilinx Virtex FPGAs. Logic block (CLB) counts, target devices, and the achieved clock speed of the mapped designs appear in Table I. Each Virtex CLB contains four LUTs. Test session counts were determined for these paths using both the single phase and multi-phase methods. Test statistics for the circuits are given in Table II.

The total time needed to test each of the benchmarks in our experiments was determined from the number of LUTs in the longest path in each test phase and the number of test sessions (Table III). Test time is determined using the method presented in Section IV-C. The total test time is based on the minimum clock cycle for each circuit (shown in Table I) and a re-configuration time per session. The re-configuration times for the XCV100 (1.2 ms), XCV300 (2.9 ms), and XCV600 (6.2 ms) were determined assuming one byte is transferred to the respective device each 66 MHz configuration clock cycle [30]. Each total test time is the sum of the re-configuration time multiplied by the number of design test sessions and the number of test clock cycles multiplied by the test cycle time. In general, test time is kept to a few seconds.

A perusal of the test sessions obtained in our experiments indicate that the most efficient test method selected depends very much on circuit structure. In most cases, Procedure 2 initially obtained multi-phase sessions with several paths tested in parallel. As the number of untargeted paths decreased, the amount of parallelism decreased. Some sessions also became single phase sessions. The last few sessions often tested individual paths.

Large numbers of paths could be tested in parallel in some of our examples. In particular, Procedure 2 generated purely parallel (i.e., single-phase) sessions for the circuits, **bheap** and **bsort**. Many of the sessions in **diffeq** and **tseng** also targeted a relatively large number of paths in parallel.

In one of our experiments (**tseng**), the total test time with the single-phase method was less than that with the multi-phase method, while the latter required fewer test sessions. This circuit has relatively long paths (14 to 16 LUTs). In this case, test application time, which is exponential in the number of LUTs on the longest path in a test session, dominates reconfiguration time and leads to longer total test time. Our multi-phase algorithm, as currently implemented, attempts to reduce the number of test sessions needed by trying to maximize the number of paths tested in each session. This could lead to a solution that has more test sessions than a single-phase one, although an optimal multi-phase solution must have no more sessions than a single-phase one. A slight modification of the method presented here can be used to obtain test sessions that reduce the test time. The use of a measure such as the total test time per path tested may lead to multi-phase solutions with lower total test time.

As a final test to validate the practicality of the approach, the test configuration circuits required to test all paths within 10% delay of the critical path delay were created for designs **alu4**, **apex2**, and **seq** and applied to a Virtex XCV100. For the multi-phase approach, the required test session totals are shown in Table II. As stated in Section V, one configuration circuit is generated per test session. For this experiment, all required test circuitry (as shown in Fig. 1) for each test session was created, combined with LUTs and programmable interconnects in the test paths, and synthesized. As described in Section V, after placement and routing with our enhanced VPR, results were input into the Xilinx XSE tools for timing performance and routing verification. For all test sessions it was found that it was possible to successfully place and route all test circuitry and interface control signals to the paths under test. Since the

Design	Source	Virtex Part	Array Size	CLBs	Wires	Speed (MHz)
bsort	RAW	XCV600E	48x72	2,815	11,204	16.1
bheap	RAW	XCV600E	48x72	2,733	10,909	23.1
alu4	MCNC	XCV100E	20x30	391	1018	39.4
apex2	MCNC	XCV100E	20x30	491	1438	33.1
clma	MCNC	XCV600E	48x72	2133	6134	14.4
diffeq	MCNC	XCV100E	20x30	379	1180	27.3
elliptic	MCNC	XCV600E	48x72	906	2450	15.0
frisc	MCNC	XCV300E	32x48	894	2280	14.2
seq	MCNC	XCV100E	20x30	457	1320	39.3
tseng	MCNC	XCV300E	32x48	166	828	29.9

TABLE I
Design statistics for benchmark circuits

Circuit	Tested Paths	Multi-phase		Single Phase
		Sessions	Phases	Phases = Sessions
bsort	101	1	1	1
bheap	295	4	4	4
alu4	80	15	56	56
apex2	27	5	13	13
clma	136	17	55	55
diffeq	482	231	263	263
elliptic	155	24	26	26
frisc	379	34	84	84
seq	150	24	89	89
tseng	4,169	993	1,758	1,377

TABLE II
Test statistics for benchmark circuits

Circuit	Multi-phase			Single Phase		
	Sessions	Test Clock Cycles	Test Time (s)	Sessions	Test Cycles	Test Time (s)
bsort	1	24	0.006	1	24	0.006
bheap	4	192	0.025	4	192	0.025
alu4	15	86,016	0.021	56	81,408	0.069
apex2	5	39,936	0.007	13	39,936	0.017
clma	17	4,184,064	0.377	55	2,758,656	0.522
diffeq	231	11,796,480	0.725	263	11,649,024	0.769
elliptic	24	9,338,880	0.764	26	9,142,272	0.764
frisc	34	77,568	0.104	84	40,320	0.246
seq	24	136,704	0.031	89	135,168	0.112
tseng	993	165,789,696	8.351	1,377	131,776,512	8.347

TABLE III
Results for single and multi-phase test

placed and routed test circuitry for all test sessions (including the counter) was successfully restricted to operate at the same clock speed as the original design, it was possible to test each path at the target clock speed.

Table IV indicates the number of paths tested during each test session and the amount of logic resources used by the test circuitry and overall (including tested logic). All tests were conducted at the clock speed of the respective design (Table I). Note the decrease in the number of paths tested during the later test sessions.

	alu4			apex2			seq		
	LUTs	FFs	Paths	LUTs	FFs	Paths	LUTs	FFs	Paths
Session	Test/ Total	Test/ Total		Test/ Total	Test/ Total		Test/ Total	Test/ Total	
1	23/49	18/27	6	21/47	19/28	8	26/64	21/36	12
2	23/56	18/26	6	21/47	18/26	6	23/57	19/30	9
3	20/51	17/24	6	21/45	18/26	6	20/52	19/27	10
4	20/50	18/28	8	21/42	17/23	4	23/59	19/31	9
5	20/52	18/27	8	23/46	18/24	3	20/47	19/31	10
6	17/37	17/22	5				23/56	19/30	9
7	20/48	17/25	6				20/48	17/25	7
8	18/41	18/25	6				23/51	19/30	9
9	17/39	16/21	4				25/50	19/27	4
10	22/41	17/22	3				20/47	17/25	6
11	17/40	17/23	5				18/39	18/23	6
12	17/36	17/23	5				18/39	18/25	6
13	17/34	16/21	4				17/35	17/23	5
14	17/37	17/23	5				18/38	18/25	6
15	17/31	15/19	3				17/35	17/23	5
16							17/35	17/23	5
17							17/31	17/22	5
18							20/41	16/22	4
19							22/40	17/21	3
20							18/39	18/24	6
21							17/31	16/21	4
22							22/41	17/23	3
23							17/35	16/21	4
24							17/29	15/19	3
total			80			27			150

TABLE IV

PER-TEST SESSION STATISTICS FOR SELECTED DESIGNS

VII. CONCLUSION

In this paper, we have presented a new approach to testing selected sets of paths in FPGA-based circuits. Our approach tests these paths for all combinations of inversions along them to guarantee that the maximum delays along the tested paths will not exceed the clock period during normal operation. While the test method requires reconfiguring the FPGA for testing, the tested paths use the same connection wires, multiplexers and internal logic connections as the original circuit, ensuring the validity of the tests. Following testing, the test circuitry is removed from the device and the original user circuit is programmed into the FPGA.

Two methods have been presented for reducing the number of test configurations needed for a given set of paths. In one method, called the single-phase method, paths are selected so that all paths in each configuration can be tested in parallel. The second method, called the multi-phase method, attempts to test the paths in a configuration with a sequence of test phases, each of which tests a set of paths in parallel. Our experimental results with benchmark circuits show that these methods are viable, but the preferable method depends on the circuit structure.

While our approach has been shown to be feasible, the algorithms presented are greedy algorithms that simply maximize the number of target paths tested in each configuration. They are by no means optimal and may not result in the smallest number of configurations or total test time. The use of other criteria, such as the total time for configuration and test application for each configuration, or better heuristics may lead to more efficient testing with the proposed approach.

REFERENCES

[1] 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 *IEEE Int. Test Conf.*, Atlantic City, NJ, Sept. 1999, pp. 28–30.

[2] M. Abramovici, C. Stroud, and J. Emmert, "Online BIST and BIST-based diagnosis of FPGA logic blocks," *IEEE Trans. on VLSI Systems*, vol. 12, no. 12, pp. 1284–1294, Dec. 2004.

[3] I. G. Harris and R. Tessier, "Interconnect testing in cluster-base FPGA architectures," in *ACM/IEEE Design Automation Conf.*, Los Angeles, CA, June 2000, pp. 49–54.

[4] I. G. Harris and R. Tessier, "Testing and diagnosis of interconnect faults in cluster-based FPGA architectures," *IEEE Trans. on CAD*, vol. 21, no. 11, pp. 1337–1343, Nov. 2002.

[5] W.K. Huang, F.J. Meyer, X-T. Chen, and F. Lombardi, "Testing configurable LUT-based FPGAs," *IEEE Trans. on VLSI Systems*, vol. 6, no. 2, pp. 276–283, June 1998.

[6] C. Stroud, S. Konala, P. Chen, and M. Abramovici, “Built-in self-test of logic blocks in FPGAs (Finally, a free lunch),” in *IEEE VLSI Test Symp.*, Princeton, NJ, Apr. 1996, pp. 387–392.

[7] C. Stroud, S. Wijesuriya, C. Hamilton, and M. Abramovici, “Built-in self-test of FPGA interconnect,” in *IEEE Int. Test Conf.*, Washington, D.C., Oct. 1998, pp. 404–411.

[8] L. Zhao, D.M.H. Walker, and F. Lombardi, “IDDQ testing of bridging faults in logic resources of reprogrammable field programmable gate arrays,” *IEEE Trans. on Computers*, vol. 47, no. 10, pp. 1136–1152, Oct. 1998.

[9] M. Renovell, J. Figuras, and Y. Zorian, “Test of RAM-based FPGA: Methodology and application to the interconnect,” in *IEEE VLSI Test Symp.*, Monterey, California, Apr. 1997, pp. 230–237.

[10] C-A. Chen and S.K. Gupta, “Design of efficient BIST test pattern generators for delay testing,” *IEEE Trans. on CAD*, vol. 15, no. 12, pp. 1568–1575, Dec. 1996.

[11] S. Pilarski and A. Pierzynska, “BIST and delay fault detection,” in *IEEE Int. Test Conf.*, Baltimore, MD, Oct. 1993, pp. 236–242.

[12] A. Krasniewski, “Application-dependent testing of FPGA delay faults,” in *Euromicro Conf.*, Milan, Italy, Sept. 1999, pp. 260–267.

[13] A. Krasniewski, “Enhancing detection of delay faults in FPGA-based circuits by transformations of LUT functions,” in *IFAC Workshop on Programmable Devices and Systems*, Ostrava, CZ, Feb. 2000, pp. 127–132.

[14] A. Krasniewski, “Exploiting reconfigurability for effective detection of delay faults in LUT-based FPGAs,” in *Int. Conf. on Field Programmable Logic and Applications*, Villach, Austria, Aug. 2000, pp. 675–684.

[15] A. Krasniewski, “Evaluation of delay fault testability of LUTs for the enhancement of application-dependent testing of FPGAs,” *Journal of Systems Architecture*, vol. 49, no. 4, pp. 283–296, Sept. 2003.

[16] M. Abramovici and C. Stroud, “BIST-based delay-fault testing in FPGAs,” *Journal of Electronic Testing*, vol. 19, no. 5, pp. 549–558, Oct. 2003.

[17] E. Chmelar, “FPGA interconnect delay fault testing,” in *IEEE Int. Test Conf.*, Charlotte, NC, Sept. 2003, pp. 1239–1247.

[18] M. Tahoori and S. Mitra, “Interconnect delay testing of designs on programmable logic devices,” in *IEEE Int. Test Conf.*, Charlotte, NC, Oct. 2004.

[19] I. G. Harris, P. R. Menon, and R. Tessier, “BIST-based delay path testing in FPGA architectures,” in *IEEE Int. Test Conf.*, Baltimore, MD, Nov. 2001, pp. 932–938.

[20] L.T. Wang and E.J. McCluskey, “Complete feedback shift register design for built-in self-test,” in *Int. Conf. Comput.-Aided Design*, Santa Clara, CA, Oct. 1986, pp. 56–59.

[21] E.J. McCluskey, *Logic Design Principles with Emphasis on Testable Semi-custom Circuits*, Prentice-Hall, Inc., Upper Saddle River, NJ, 1991.

[22] P. Girard, O. Heron, S. Pravossoudovitch, and M. Renovell, “Defect analysis for delay-fault BIST in FPGAs,” in *IEEE Int. On-Line Testing Symp.*, Kos Island, Greece, July 2003, pp. 124–128.

[23] A. Krasniewski, “Evaluation of testability of path delay faults for user-configured programmable devices,” in *Int. Conf. on Field Programmable Logic and Applications*, Lisbon, Portugal, Sept. 2003, pp. 828–838.

[24] S. Padmanaban and S. Tragoudas, “Efficient identification of (critical) testable path delay faults using decision diagrams,” *IEEE Trans. on CAD*, vol. 24, no. 1, pp. 77–87, Jan. 2005.

[25] E. S. Park and M. R. Mercer, “An efficient delay test generation system for combinational logic circuits,” *IEEE Trans. on CAD*, vol. 11, no. 7, pp. 926–938, July 1992.

[26] W. Xu, R. Ramanarayanan, and R. Tessier, “Adaptive fault tolerance for networked reconfigurable systems,” in *IEEE Symp. Field-Programmable Custom Computing Machines*, Napa, CA, Apr. 2003, pp. 143–152.

[27] V. Betz and J. Rose, “VPR: a new packing, placement, and routing tool for FPGA research,” in *Int. Conf. on Field Programmable Logic and Applications*, Oxford, UK, Sept. 1997, pp. 213–222.

[28] S. Yang, “Logic synthesis and optimization benchmarks user guide, version 3.0,” Tech. Rep., Microelectronics Centre of North Carolina, 1991.

[29] Jonathan Babb, Matthew Frank, Victor Lee, Elliot Waingold, and Rajeev Barua, “The RAW benchmark suite: Computation structures for general purpose computing,” in *IEEE Workshop on FPGA-based Custom Computing Machines*, Napa, Ca, Apr. 1997, pp. 134–144.

[30] *Virtex Data Sheet*, Xilinx Corporation, 2002.

PLACE PHOTO HERE

Premachandran R. Menon received his B.Sc. from the Banaras Hindu University, Varanasi, India in 1954 and the Ph.D. from the University of Washington, Seattle, WA in 1963, both in Electrical Engineering. He is now a retired professor in the Department of Electrical and Computer Engineering at the University of Massachusetts, Amherst, MA. Prior to joining the University, he was with Bell Laboratories, first at Murray Hill, NJ and later at Naperville, IL where he was a Distinguished Member of Technical Staff. He has served on the Editorial Boards of the *IEEE Transactions on Computers* and the *Journal of Design Automation and Fault Tolerant Computing*. His research interests are VLSI circuit testing and testable design.

PLACE PHOTO HERE

Weifeng Xu received the B.S. and M.S. degrees in electrical engineering from Fudan University, Shanghai, China in 1997 and 2000, respectively. He is current pursuing the Ph.D. degree at the University of Massachusetts, Amherst. His research interests include reconfigurable computing and fault tolerant systems.

PLACE PHOTO HERE

Russell Tessier is an associate professor of electrical and computer engineering at the University of Massachusetts, Amherst, MA. He received the B.S. degree in computer engineering from Rensselaer Polytechnic Institute, Troy, NY in 1989 and S.M. and Ph.D. degrees in electrical engineering from MIT, Cambridge, MA in 1992 and 1999, respectively. Dr. Tessier is a founder of Virtual Machine Works, a logic emulation company, and has also worked at BBN, Ikos Systems, and Altera. Prof. Tessier currently leads the Reconfigurable Computing Group at UMass. His research interests include computer architecture, field-programmable gate arrays, and system verification.