BIST-Based Delay-Fault Testing in FPGAs

Miron Abramovici
DAFCA
Berkeley Heights, NJ
miron@dafca.com

Charles E. Stroud
Electrical and Computer Engineering, Auburn University
Auburn, AL
cestroud@eng.auburn.edu

Keywords: Built-In Self-Test, Field Programmable Gate Arrays, Delay Faults

Abstract: We present the first delay-fault testing approach for Field Programmable Gate Arrays (FPGAs), applicable for on-line testing as well as for off-line manufacturing and system-level testing. Our approach is based on Built-In Self-Test (BIST), it is comprehensive, and does not require expensive external test equipment (ATE). We have successfully implemented this BIST approach for delay-fault testing on the Lattice ORCA 2C and Xilinx Spartan FPGAs.

1. Introduction

Advances in VLSI technology have resulted in more defects affecting the delays in the circuit, thus increasing the importance of delay-fault testing. The current FPGA manufacturing testing practice relies on configuring the FPGA with many designs and running them at speed. This is useful for speed-binning, but it may not be a comprehensive delay-fault test. One of the main difficulties in the problem of delay-fault testing for FPGAs originates in the fact that users' circuits are not known when the chip is manufactured. While for an Application-Specific Integrated Circuit (ASIC) the target clock frequency is known before the chip is fabricated, it is impossible for an FPGA manufacturer to test the circuits that will be implemented on each device. Previous work in FPGA delay-fault testing [9][10] considered only the testing of the user's circuit. However, such an approach is problematic both for FPGA manufacturers, who want to ship defect-free chips, and for users, who want to purchase defect-free parts (otherwise, system-level debugging becomes very difficult if design problems are intermixed with manufacturing faults). Testing only the user's circuit is also unsafe for on-line testing, because new faults are equally likely to occur in spare resources or in the currently unused portion of the operational part of the system. These dormant faults may accumulate and have a detrimental effect on the system reliability since these spare and unused resources will be used in fault-tolerant applications to replace defective resources [15]. Thus to guarantee a reliable operation, an on-line test must completely check all the system resources, including spares.

In testing FPGAs, we can take advantage of their unique properties - reconfigurability, partial reconfigurability, and regular structure - to achieve features that cannot be realized in ASIC testing. Unlike in ASICs, where BIST requires large area overhead and some performance degradation, FPGA BIST can be done with no area overhead and no delay penalty [16]. In FPGAs, we can pseudo-exhaustively test both the *programmable logic blocks* (PLBs) and the programmable interconnect network, during both off-line [4][18][19] and on-line testing [3][17] with reasonable test application times. Pseudoexhaustive testing [12] is practically complete and removes the need to evaluate the test quality. Thus, pseudoexhaustive FPGA BIST approaches do not require either Automatic Test Pattern Generation (ATPG) or fault simulation, which are computationally expensive tasks for ASICs. Accurate fault diagnosis is difficult to achieve in ASICs, but it can be done efficiently in FPGAs [2][4][18]. In ASICs, fault tolerance for manufacturing faults cannot be accomplished without massive redundancy, but FPGAs allow efficient fault tolerance without dedicated spare resources as a result of the ability to reconfigure around faults [5][6].

Our first attempt to perform FPGA delay-fault testing was to run our BIST configurations for logic and interconnect at the specified clock rate for the target FPGA. However, we quickly learned that this approach cannot work without significantly increasing the number of BIST configurations that must be applied to the FPGA. The number of configurations is the dominant factor determining the total test time, because the configuration download time is several orders of magnitude larger than the test-pattern application time. To minimize the total number of configurations, our BIST techniques try to test as many resources as possible within the same configuration. For off-line PLB test, this requires distributing the patterns from the test-pattern generator (TPG) to many PLBs under test using long signal paths. Similarly, for interconnect test, the wires under test are long, connecting many wire segments and programmable switches. Under these conditions, we must run the BIST configurations at a frequency much lower than that used in normal operation, so most delay-faults will not be detected by these tests. An alternative would be to reduce the amount of logic and/or interconnect that is under test during any given BIST configuration to allow BIST execution at a much higher clock frequency; however, this will require many more configurations, and ultimately significant increases in testing time and cost [18].

In this paper, we introduce a novel BIST-based technique for delay-fault testing in FPGAs. Our technique is independent of the system applications implemented on the FPGA, and it is applicable for both on-line testing (in the context of the Roving STARs approach [3]) and for off-line manufacturing and system-level testing. Our method is based on BIST, it is comprehensive, and it can work with any low-cost ATE. We have successfully implemented our approach on two different FPGAs - ORCA 2C from Lattice [11] and Spartan from Xilinx [20].

The remainder of this paper is organized as follows. In Section 2 we review the basic structure of an FPGA. In Section 3 we present the principle of the BIST technique, and in Section 4 we analyze its implementation issues. In Section 5 and Section 6 we discuss the application of the delay-fault BIST approach to on-line and off-line testing, respectively. An implementation example using a Spartan FPGA is presented in Section 7 and we conclude the paper in Section 8.

2. The FPGA Structure

An FPGA is a two-dimensional array of PLBs, interfacing to its Input/Output (I/O) pins via programmable I/O cells. Communication among PLBs and I/O cells is done through a programmable interconnect network, consisting of wire segments that can be connected via programmable switches referred to as *configurable interconnect points* (CIPs). The PLB logic functions and the CIPs are controlled by writing the configuration RAM. Wire segments in the programmable interconnect network are bounded by these CIPs and are considered to be either global or local routing resources. Global routing resources connect non-adjacent PLBs, while local routing resources connect a PLB to global routing resources or to adjacent PLBs. The routing resources are bus-oriented, with the number of wires per bus typically ranging between 4 and 8.

The basic CIP structure consists of a transmission gate controlled by a configuration memory bit (Figure 1a). There are three types of CIPs which we refer to as the *cross-point CIP* (Figure 1b), the *break-point CIP* (Figure 1c), and the *multiplexer* (*MUX) CIP* (Figure 1d) [11]. While a cross-point CIP connects wire segments located in disjoint planes (a horizontal segment with a vertical one), a break-point CIP connects two wire segments in the same plane. The MUX CIP comes in two varieties: decoded and non-decoded. A decoded MUX CIP is a group of 2^k cross-point CIPs sharing a common output wire and controlled by k configuration bits, such that the input wire being addressed by the configuration bits is connected to the output wire; the decoding logic is incorporated between the configuration bits and the transmission gates. A non-decoded MUX CIP contains a configuration bit for each transmission gate, such that k wire segments are controlled by k configuration bits; only one of the configuration bits can be active for any given configuration. There is also a *compound CIP* (Figure 1e), which is a combination of four cross-point and two break-point CIPs, each separately controlled by a configuration bit [20]. Most recent FPGA interconnect architectures are primarily constructed from non-decoded MUX CIPs that are buffered to prevent signal degradation due to the series resistance of each transmission gate along the signal path. A signal path is formed by connecting several wire segments and PLBs in a continuous sequence via multiple CIPs. The propagation delay along the path accumulates the delays of all its PLBs, wire segments, and CIPs. A path may have different delays for rising (0/1) and falling (1/0) transitions.

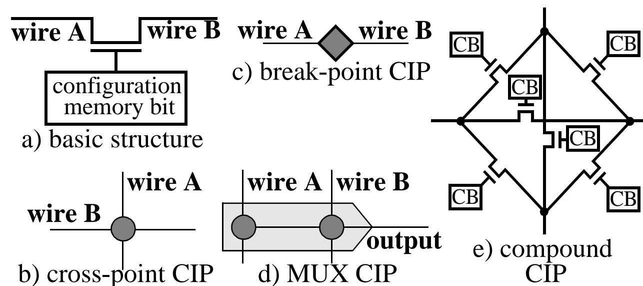


Figure 1. Configurable Interconnect Points

Figure 2 illustrates the typical structure of a PLB, consisting of small RAMs used as look-up tables (LUTs), flip-flops (FFs) that can also be configured as latches, and output MUX logic. Often the RAMs can also be operated as writable memories. The LUTs can also implement special functions such as adders or multipliers.

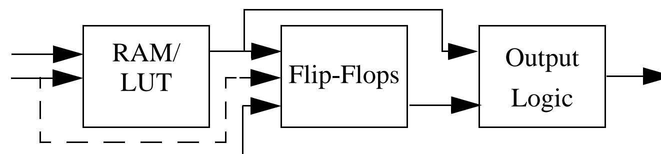


Figure 2. Typical PLB structure

3. BIST for Delay-Fault Testing

3.1 The Main Idea

Figure 3a outlines our delay-fault BIST architecture. We configure several *paths under test* (PUTs), so that every path has the same sequence of PLBs, wire segments, and CIPs. Each PLB on the path is programmed as an identity function so that it appears as a buffer for the signal propagating along the path. The PUTs are identical, except for their position in the FPGA, so that their propagation delays will be about the same in the fault-free case. This works well with the bus structure of the programmable interconnect of most FPGAs.

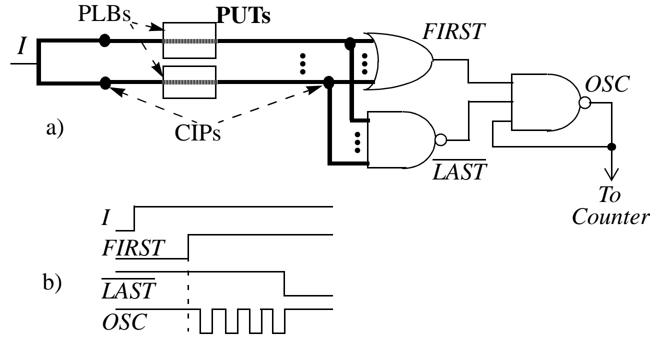


Figure 3. The basic principle

The basic idea of the BIST technique is to compare the delays of the PUTs. Assume that a rising transition is applied at their common input *I*. This transition propagates along every PUT, and it will eventually appear at the inputs of the OR and the NAND gates. The signal *FIRST* responds to the fastest arriving transition, while $\overline{LAST}$ changes only after the slowest one has arrived. *FIRST* enables the local oscillator loop, and $\overline{LAST}$ stops the oscillations (see Figure 3b). Thus the count of *OSC* pulses measures the difference *D* between the fastest and the slowest propagation delays along the PUTs. In a circuit free of delay-faults, *D* should be smaller than a predetermined threshold; otherwise we say that a delay-fault is detected. Note that the same circuit can detect a delay-fault affecting the propagation of a 1/0 transition, the only difference being that the roles of *FIRST* and $\overline{LAST}$ are reversed.

Since the first *OSC* pulse may be generated (possibly as a partial pulse) even when the transitions of *FIRST* and $\overline{LAST}$ are very close, a count of one should *not* be interpreted as indicating a delay-fault.

3.2 Testing PLB Delays

Figure 4 shows the structure of a PUT traversing both a LUT and a FF inside a PLB. The raising input transition is applied to all LUT inputs, and the LUT is configured as an AND gate, whose output propagates the slowest of its input transitions. The FF/latch is configured as a latch, and its clock input is kept at the active value, so that the latch will be in the transparent mode and will behave like a buffer. In this way the entire PLB implements an identity function. The paths bypassing the FFs and the paths bypassing the LUTs are tested by similar configurations. When propagating a falling transition, the LUT is configured to implement an OR gate.

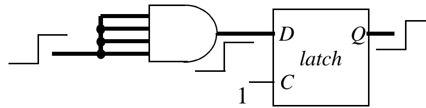


Figure 4. PUT traversing a PLB

It is interesting to observe that, unlike ASIC delay-fault testing, this technique does not involve clocking using the system clock. As a result, the clock distribution network in the FPGA is not tested for delay-faults using this technique. This is not a problem, since delays on the clock distribution paths are implicitly checked during speed-binning tests. Thus our delay-fault BIST should be done in addition to, and not as a replacement of, speed-binning.

3.3 Complete Testing of LUT Paths

In any addressing/multiplexing mechanism with *k* address bits, there are 2^k paths from every address input to the output, and each one of the 2^k input combinations sensitizes a different set of *k* paths. As described so far, our technique applies only the all-0 and all-1 input vectors. A complete delay-fault test of the LUT must apply every possible address *i* to the LUT inputs, with the LUT programmed to produce a transition when the inputs change to the target address *i*; for every target address, the LUT should generate once a 0/1 and once a 1/0 transition. One simple way to generate a 0/1 transition is to program a 1 at the address *i* and 0 at all other addresses. Then every input change from any other address to *i* will create a 0/1 spike-free output transition,

occurring in response to the slowest input-output propagation through the LUT. Similarly, programming the LUT with a 0 at the target address and 1 elsewhere will generate a spike-free 1/0 transition.

Figure 5 illustrates one BIST configuration applying this technique. Here LUTs have k=2 inputs, and the PUTs connect only LUTs, bypassing FFs in every PLB. Inside every LUT we indicate the target address for this configuration. A LUT without (with) an inverting bubble is programmed to generate a 0/1 (1/0) transition. The PUTs traverse consecutive groups of 2^k=4 pairs of LUTs, where every group has the same configuration (only one such group is shown in Figure 5). Every pair in a group has a different target address, which corresponds with the final values of the input transitions for that pair. Note that the pattern for programming either a 0 or a 1 in the target address for a given LUT is a function of the target address for the subsequent LUT in the PUT.

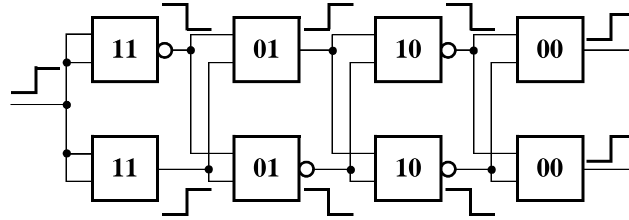


Figure 5. PUTs for LUTs

For every LUT, the configuration of Figure 5 checks only one address and only one transition. Similar configurations are easily constructed so that every LUT generates both 0/1 and 1/0 transitions for every target address. The total number of configurations needed for a complete test is 2^{k+1} configurations for an k-input LUT. Since k is typically small (3 or 4) for most LUTs, the total number of test configurations is not prohibitive.

3.4 Testing Delays for Adder Configurations

Other modes of operation of a PLB, such as an adder, may involve dedicated logic and dedicated interconnect resources whose delays can be tested only when the PLB is configured for these operations. Figure 6 illustrates the PUTs created for testing delays through a PLB configured as an adder that computes the k-bit sum (S) of two k-bit inputs A and B; C_{in} is the carry-in and C_{out} is the carry-out. Note that the sum logic of the adder is implemented by the LUTs with no dedicated logic. As a result, the delays of the $A \rightarrow S$ and $B \rightarrow S$ paths are tested with configurations of the type shown in Figure 4. Here we test only the delays associated with the paths from C_{in} and the paths to C_{out}, and also with the inter-PLB dedicated carry routing typically found in most FPGAs.

In Figure 6, **0** (**1**) denotes a k-bit all-0 (all-1) vector. When A=**0**, C_{out} implements the AND function of C_{in} and all the B inputs (this is a functional property independent of the implementation of the adder). In Figure 6a, we set A=**0** and apply a raising transition to C_{in} and every B input. Then C_{out} undergoes a raising transition only after the slowest propagation of the raising input transition along the $C_{in} \rightarrow C_{out}$ and $B \rightarrow C_{out}$ paths completes. The PUT is formed by connecting C_{out} of one PLB to the C_{in} and B inputs of an adjacent PLB which is identically configured. Note that the PUT is using the dedicated carry-chain connections between PLBs. This repetitive structure is a form of iterative logic array.

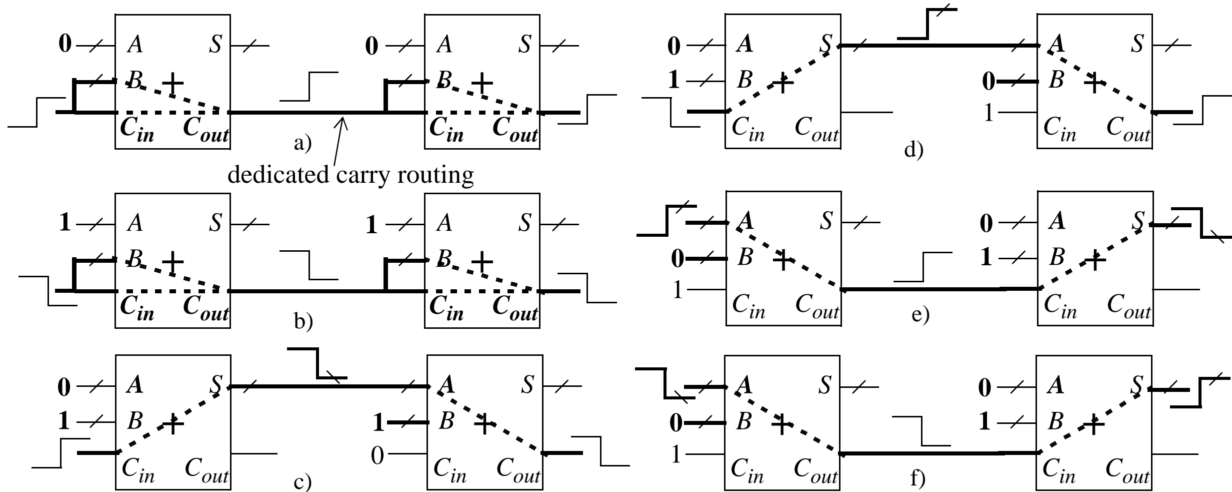


Figure 6. PUTs for testing carry paths in an adder

We reuse the same configuration to test the propagation of a falling transition (Figure 6b) by setting A=**1**, which makes C_{out} implement the OR function of C_{in} and all the B inputs. Then C_{out} undergoes a falling transition only after the slowest propagation of the falling input transition through the PLB completes.

The other parts of Figure 6 illustrate the testing of the $C_{in}\rightarrow S$ and $A\rightarrow C_{out}$ paths. In Figure 6c, setting A=**0** and B=**1** for the first PLB enables the propagation of the raising transition applied on C_{in} to every S signal, where it appears as a falling transition. The S signals from the first PLB are connected to the A inputs of the second PLB, where, because B=**1**, C_{out} implements the OR function of all the A inputs (here C_{in}=0). Each PUT combines a $C_{in}\rightarrow S$ path in the first PLB with a $A\rightarrow C_{out}$ path in the second one. To further process the falling transition from C_{out} of the second PLB, the next two PLBs on the PUT are set up as shown in Figure 6d. This configuration tests the $C_{in}\rightarrow S$ paths for a raising transition in the first PLB and for a falling transition in the third PLB, and tests the $A\rightarrow C_{out}$ paths for a falling transition in the second PLB and for a raising transition in the fourth PLB.

We reuse the same configuration to test the propagation of a falling transition; for this, the first group of two PLBs is set up as in Figure 6d, and the second group as in Figure 6c. Figure 6e and Figure 6f illustrate another configuration, where the roles of the two PLBs in a group are interchanged, so that the $A\rightarrow C_{out}$ paths are tested in the first and the $C_{in}\rightarrow S$ paths in the second one. Like before, the PUT goes through two groups of two PLBs (Figure 6e and Figure 6f). To test the same paths for opposite transitions, the first group is set up as shown in Figure 6f, and the second as in Figure 6e.

Note that three configurations are sufficient to test all the carry paths, independent of the size k of the adder. The ability to test all carry paths is essential to the testing of dedicated carry logic and routing resources.

4. Implementation Issues

The delay-fault BIST circuitry is very simple: the TPG only needs to generate the two transitions, and the *output response analyzer* (ORA) consists of the three gates that produce the oscillation and the counter. The counter is reset before each experiment. Both the TPG and the counter can be initialized, and the ORA counter results can be read, via the FPGA boundary-scan access mechanism; this is the preferable method for on-line testing. Alternatively, for off-line testing, the ORA counter results can be read via configuration memory readback with the TPG and counter initialized via a global reset following download of the BIST configuration.

The smallest difference between the delay of the fastest and slowest PUTs detectable with our scheme corresponds to one *OSC* cycle. When testing a path with ASIC-type delay-fault testing, the smallest detectable delay-fault is generally about 5% of the path delay. To achieve a similar feature, PUTs should be constructed so that their total propagation delay corresponds to at least 20 *OSC* cycles.

While making PUTs as long as possible would increase the number of FPGA resources concurrently tested, and possibly reduce the total number of BIST configurations required for a complete delay-fault test, it may also cause false negative results. For example, assume a path *P1* where all of its components (PLBs, CIPs, and wire segments) are just 1% slower than their counterparts on path *P2*. If the PUTs involve a large number of components, the accumulated difference between the delays of *P1* and *P2* may be incorrectly reported as a delay-fault. Therefore, PUTs should be constructed so that their delay is not significantly larger than that of an average path that would be used in "normal" system circuits implemented in the FPGA while, at the same time being large enough to obtain the desired delay-fault detection resolution (for example, the 20 *OSC* cycles described above).

In any comparison-based BIST approach, a passing result may be produced when the compared elements are all faulty. In our case, this means that all the compared PUTs are equally slow. Such a situation is unlikely when we compare several (4 to 8) paths. However, if desired, a validation test to protect against this case can be easily done by selecting one of the paths that passed the test and comparing it with a new path which was not part of the compared group that passed the initial test.

No delay-faults will be detected in a slow device where all paths are equally slow. This is the correct result, and such a chip will be identified by speed-binning and may be allowed to work as a lower speed-grade device.

Our approach may fail if a PUT has compensating delay-faults, where the detection of a slow path segment is masked by the presence of a fast segment, so that the overall path delay remains about the same as the other PUTs. In general, however, most delay-faults slow down the circuit, and such a multiple fault is unlikely to occur in practice.

Based on these arguments we can conclude that, if we include each resource that can contribute to a delay-fault in one of the PUTs, and we test each PUT for both rising and falling transitions, ***our delay-fault test is complete***, that is, it will detect any delay-fault that creates a meaningful difference between compared PUTs. Our path selection follows the scheme used in our interconnect testing approach [19], and guarantees that every resource in included (at least once) in a PUT. Hence we do not need to compute the resulting delay-fault coverage.

The use of the local oscillator created from the inverting feedback in the PLB logic could give rise to concerns of the quality of the clock feeding the ORA counter, specifically, the duty cycle and period needed for proper operation of the counter. One solution to this problem is to configure a single FF as a toggle FF with the output of the local oscillator driving the clock input to this FF, and the output of the toggle FF driving the clock input of the ORA counter. This effectively divides the local oscillator frequency by 2 and ensures a near 50% duty cycle to the ORA counter. The lower frequency clock will only reduce the resolu-

tion of delay-fault detection as opposed to preventing this delay-fault BIST approach from working. However, we have implemented the delay-fault BIST approach in an ORCA 2C15A FPGA and found the oscillator clock to run at 243 MHz while producing a duty cycle and clock waveform of sufficient quality to obtain reproducible results from one execution of the delay-fault BIST sequence to the next. Therefore, dividing the clock may not be necessary.

Unlike in ASICs, where delays may be affected by crosstalk or other deep-submicron effects, in FPGAs the layout is already done, and these problems have already been solved by the device manufacturer. Thus, we can safely ignore these effects in FPGA delay-fault testing.

5. On-Line Delay-Fault Testing

Our ***roving STARs approach*** [1][3] introduced new techniques for on-line FPGA testing, diagnosis, and fault-tolerance, applicable to any FPGA supporting incremental *run-time reconfiguration* (RTR) via its boundary-scan interface [14]. A STAR (self-testing area) is a temporarily off-line section of the FPGA where self-testing occurs without disturbing the normal system activity in the rest of the chip. Roving the STARs periodically brings every section of the FPGA under test. Our logic [2] and interconnect [17] BIST approaches guarantee complete testing of the FPGA, including all its spare resources, and do not require any part of the chip to be fault-free. The same holds true for our delay-fault BIST approach.

Figure 7 depicts an FPGA with a vertical STAR (V-STAR) and an horizontal STAR (H-STAR); the system application resides in the working areas outside the STARs. V-STAR is two-column wide, and H-STAR is two-row wide. Note that global horizontal routing resources in V-STAR and global vertical routing resources in H-STAR may be used by the system signals connecting the working areas separated by the STARs. Partial RTR via the boundary-scan interface allows the test configurations used by STARs to be downloaded without impacting the system operation. After self-testing of a STAR has been completed (both for PLBs and interconnect), the STAR roves to a new location, by exchanging places with an equal-size slice of the working area; roving the STARs across the FPGA is implemented by a sequence of precomputed partial reconfigurations and assures that the entire FPGA will be eventually tested. The roving process and the use of roving STARs for test and diagnosis of PLBs are described in detail in [1] and [3].

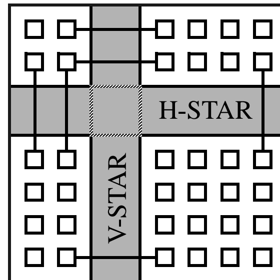
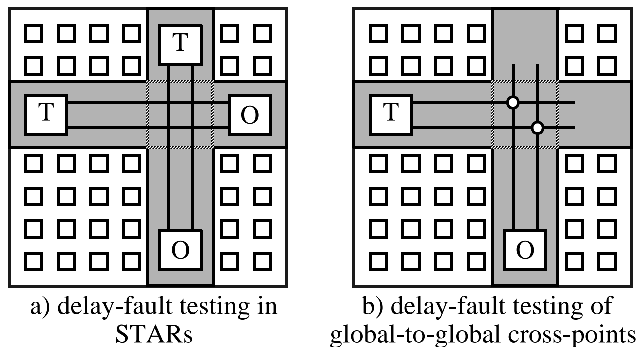


Figure 7. STARs in FPGA

Testing for delay-faults follows the pattern of interconnect testing in our on-line routing BIST [17], where horizontal and vertical routing resources are tested in H-STAR and V-STAR, respectively. Testing for delay-faults takes place after completing the test for logic and interconnect resources within the STAR. Figure 8a illustrates this process, where PUTs are fed by the TPG *T* and compared by the ORA *O*. The PUTs in H-STAR are constructed from horizontal wire segments, and the paths tested in V-STAR from vertical wire segments. Since PUTs include PLBs, delay-faults in the PLBs and in the local interconnect along PUTs are also tested. Testing for delay-faults in the cross-point CIPs connecting global horizontal and vertical routing busses must involve both STARs and can only be performed at the intersection of the two STARs, as illustrated in Figure 8b.



a) delay-fault testing in STARs

b) delay-fault testing of global-to-global cross-points

Figure 8. On-line delay-fault testing using STARs

Table 1 summarizes the set of on-line BIST configurations needed for a complete delay-fault test of a Lattice ORCA 2C FPGA in terms of the number of configurations (also called test phases) that must be downloaded in each STAR position. Note that this includes a single BIST configuration for both 0/1 and 1/0 transitions tests and complete delay-fault test of the LUTs. In general, the number of delay-fault BIST configurations for the programmable routing resources is approximately equal to the number of BIST configurations for interconnect testing given in [18]. Delay-faults in PLB logic, including dedicated carry logic and routing for adders, are tested in the V-STAR using 12 additional BIST phases in test session 2 (although the PLB testing can be performed in either STAR). Complete delay-fault testing in the LUTs requires an additional 32 BIST configurations for the 4-input LUTs of the ORCA 2C FPGA. Since some of the PLBs are programmed as ORAs to produce *OSC* and to count the number of *OSC* cycles, the 32 configurations must be applied to these PLBs to test their LUTs while previously tested PLBs are programmed as ORAs. This brings the total number of LUT BIST configurations to 64 in test session 6 which can be performed in either STAR. Therefore, a total of 117 delay-fault BIST configurations are required for complete delay-fault testing of all routing and logic resources.

Table 1. Summary of delay-fault BIST test sessions

Test Session	Target Faults	V-STAR Phases	H-STAR Phases
1	global routing	7	7
2	local routing & PLB logic	16	4
3	global-to-local interconnections	3	3
4	multiplexer CIPs & PLB logic	7	0
5	cross-point CIPs between global busses	6	
6	LUTs	64	

6. Off-Line Delay-Fault Testing

One way of characterizing the difference between on-line and off-line (manufacturing) testing is that no system function exists during off-line testing. Hence for off-line testing, we can populate the entire FPGA with a “galaxy” of parallel STARs (either vertical or horizontal), all executing concurrently the same delay-fault BIST configurations (Figure 9a shows a “galaxy” of H-STARs). A similar arrangement is used for parallel V-STARs. Since both STARs are needed for delay-fault testing of global-to-global cross-point CIPs, parallel BIST structures illustrated in Figure 9b are used. The set of BIST configurations given in Table 1 is the same for both on-line and off-line testing, requiring a total of 117 BIST configurations for complete delay-fault testing in the ORCA 2C series FPGA.

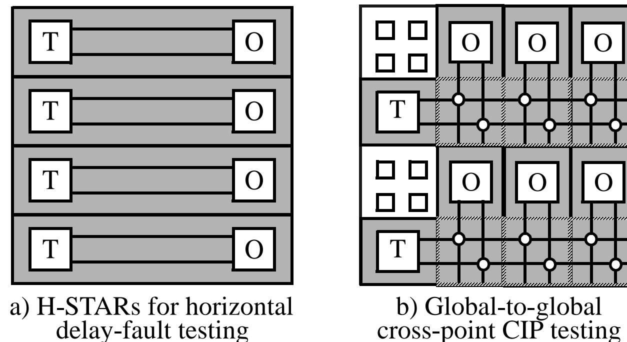


Figure 9. Galaxy delay-fault testing

The number of BIST configurations is independent of the size of the FPGA, as is the time required for the execution of the BIST sequence. However, since the dominant factor in testing time is the configuration download time, the total testing time is not independent of the size of the FPGA. For the ORCA 2C15 FPGA (a 20-by-20 array of PLBs), approximately 225,000 bits of configuration data must be downloaded for each off-line BIST configuration. This corresponds to 225,000 TCK clock cycles when downloading through the boundary-scan interface compared to only about 100 TCK cycles for execution of the BIST sequence and retrieval of the BIST results. Therefore, the BIST execution time is insignificant compared to the download time. A total of about 26,325,000 configuration bits must be downloaded for the complete set of 117 delay-fault BIST configurations with the ORA count results read at the end of each BIST sequence. At a 20MHz maximum clock frequency for TCK, this corresponds to approximately 1.3 seconds to perform all off-line delay-fault testing in the ORCA 2C15.

7. Implementation Example

We have also implemented the approach in Xilinx Spartan series FPGAs. Figure 10a shows the delay-fault BIST implementation as it would reside in a STAR taken from the Xilinx FPGA Editor [20]. The TPG is at the top of the STAR, and the ORA logic (OR and NAND gates to create *OSC*) at the bottom. The ORA counter is a 6-bit counter constructed from three PLBs implementing a 2-bit counter each. The *OSC* output drives the clock inputs of the ORA counter (Figure 10c). This implementation is for off-line testing, as the global reset generated after configuration is used to initiate the BIST sequence. The global reset cannot be used during on-line testing, since it would reset the system function; for on-line testing, the BIST sequence is initiated via the boundary-scan interface. The TPG is constructed from a shift register with the input to the shift register tied to a logic 1 and the FFs clocked by the internal 8MHz oscillator (Figure 10b). The role of the 2-bit shift register is to allow some

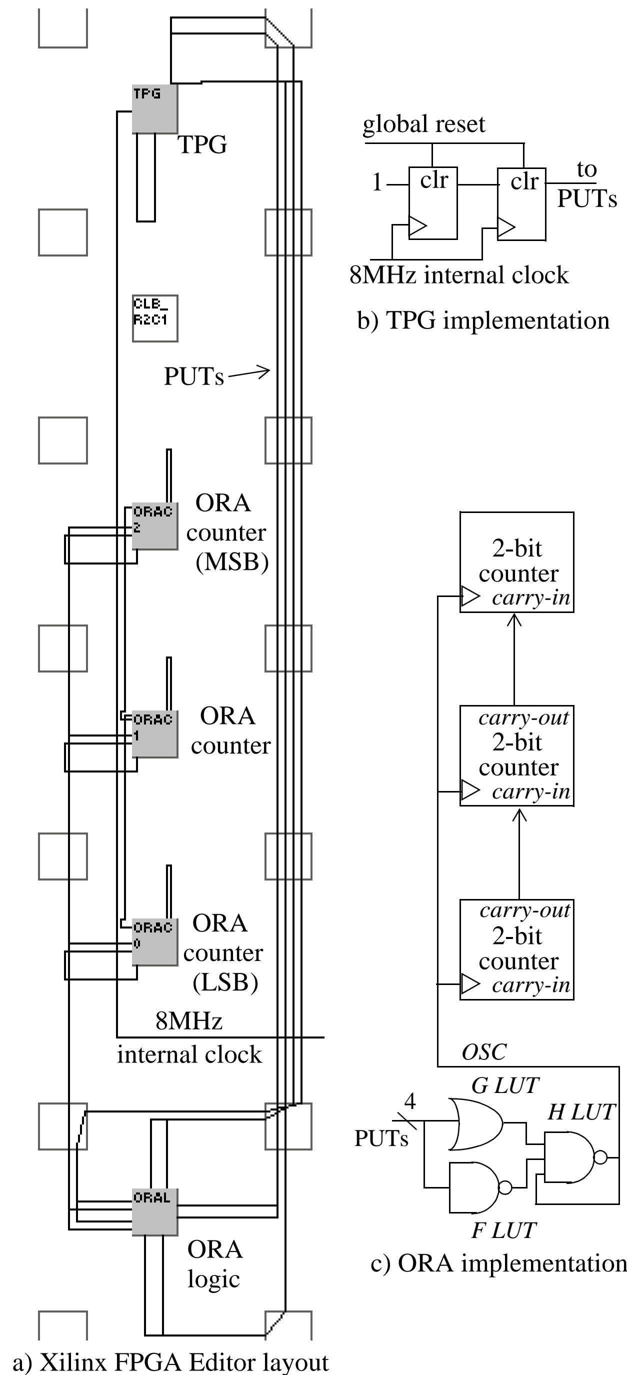


Figure 10. Delay-fault implementation in Spartan FPGA

time for the FPGA to stabilize after downloading the configuration before measuring delays; this was a precautionary measure and may not be necessary. Once configuration of the FPGA is complete, the global reset sets the two FFs to logic 0, and two 8MHz clock cycles later a 0/1 transition will start propagating on the four PUTs. The PUTs travel through an equal number of CIPs, including the switch boxes, prior to entering the ORA logic. At the conclusion of the BIST sequence, the contents of the ORA counter FFs are obtained via a configuration memory readback operation in this Spartan implementation, as opposed to the scan-chain-based boundary-scan access we used for the ORCA FPGA.

8. Conclusions

We have presented the first delay-fault testing approach for FPGAs, applicable both for off-line manufacturing and system-level testing, as well as for on-line testing within the framework of the roving STARs approach. Prior work dealt only with testing the user circuit implemented in the FPGA. Our method is based on BIST, it is comprehensive, and does not require expensive ATE. We have successfully implemented this BIST approach on the ORCA 2C and Xilinx Spartan FPGAs and have verified that the approach is not only feasible but is also practical.

We have emulated many delay-faults by creating a "faulty" PUT longer than the other "fault-free" PUTs, (the longer PUT is routed through additional wire segments, CIPs, and PLBs). In all cases, our technique successfully detected all emulated delay-faults.

The current diagnostic resolution for delay-faults detected using this approach is to a STAR. The focus of our future research will be improved diagnostic resolution in order to support efficient fault and defect tolerance for delay-faults.

Acknowledgements: We had helpful discussions with Mark Boyd on this topic. Matt Lashinsky and Thomas Slaughter of the VLSI-FPGA Design & Test Lab at UNC-Charlotte contributed to the first ORCA implementation and the first Xilinx Spartan implementation, respectively.

References

[1] M. Abramovici, C. Stroud, S. Wijesuriya, C. Hamilton, and V. Verma, "Using Roving STARs for On-Line Testing and Diagnosis of FPGAs in Fault-Tolerant Applications," *Proc. IEEE Intn'l. Test Conf.*, 1999, pp. 973-982.

[2] M. Abramovici, C. Stroud, B. Skaggs, and J. Emmert, "Improving On-Line BIST-Based Diagnosis for Roving STARs", *Proc. IEEE Intn'l. On-Line Test Workshop*, 2000, pp. 31-39.

[3] M. Abramovici, J. Emmert, and C. Stroud, "Roving STARs: An Integrated Approach to On-Line Testing, Diagnosis, and Fault Tolerance for FPGAs in Adaptive Computing Systems," *Proc. Third NASA/DoD Workshop on Evolvable Hardware*, 2001, pp. 73-92.

[4] M. Abramovici and C. Stroud, "BIST-Based Test and Diagnosis of FPGA Logic Blocks," *IEEE Trans. on VLSI*, Vol. 9, No. 1, pp. 159-172, Feb., 2001.

[5] J. Emmert, C. Stroud, B. Skaggs, and M. Abramovici, "Dynamic Fault Tolerance in FPGAs via Partial Reconfiguration," *Proc. IEEE Symp. on Field-Programmable Custom Computing Machines*, 2000, pp. 165-174.

[6] J. Emmert, S. Baumgart, P. Kataria, A. Taylor, C. Stroud, M. Abramovici, "On-line Fault Tolerance for FPGA Interconnect with Roving STARs," *IEEE Intn'l. Symp. on Defect and Fault Tolerance in VLSI Systems*, 2001, pp. 445-454.

[7] I. Harris and R. Tessier, "Interconnect Testing in Cluster-Based FPGA Architectures", *Proc. AMC/IEEE Design Automation Conf.*, 2000, pp. 49-54.

[8] I. Harris and R. Tessier, "Diagnosis of Interconnect Faults in Cluster-Based FPGA Architectures", *Proc. IEEE Intn'l Conf. on Computer Aided Design*, 2000, pp. 472-476.

[9] I. Harris, P. Menon, R. Tessier, "BIST-based Delay-Path Testing in FPGA Architectures," *Proc. IEEE Intn'l. Test Conf.*, 2001, pp. 932-938.

[10] A. Krasniewski, "Testing FPGA delay-faults in the System Environment is very Different from Ordinary delay-fault Testing," *Proc. IEEE Intn'l On-Line Test Workshop*, 2001, pp. 37-40.

[11] Lattice Semiconductor Co., *http://www.latticesemi.com/products/fpga.*

[12] E. McCluskey, "Verification Testing - A Pseudoexhaustive Test Technique," *IEEE Trans. on Computers*, Vol. C-33, No. 6, pp. 541-546, June, 1984.

[13] H. Michinishi, T. Yokohira, T. Okamoto, T. Inoue, and H. Fujiwara, "A Test Methodology for Interconnect Structures of LUT-Based FPGAs," *Proc. IEEE Asian Test Symp.*, 1996, pp. 68-74.

[14] "Standard Test Access Port and Boundary-Scan Architecture," *IEEE Standard P1149.1*, 1990.

[15] A. Steininger and S. Scherrer, "On the Necessity of On-Line BIST in Safety Critical Applications," *Proc. 29th Fault-Tolerant Computing Symp.*, 1999, pp. 208-215.

[16] C. Stroud, S. Konala, P. Chen, and M. Abramovici, "Built-In Self-Test for Programmable Logic Blocks in FPGAs (Finally, A Free Lunch: BIST Without Overhead!)," *Proc. IEEE VLSI Test Symp.*, 1996, pp. 387-392.

[17] C. Stroud, M. Lashinsky, J. Nall, J. Emmert, and M. Abramovici, "On-Line BIST and Diagnosis of FPGA Interconnect Using Roving STARs," *Proc. IEEE Intn'l. On-Line Test Workshop*, 2001, pp. 31-39.

[18] C. Stroud, J. Nall, M. Lashinsky, and M. Abramovici, "BIST-Based Diagnosis of FPGA Interconnect," *Proc. IEEE Intn'l Test Conf.*, 2002, pp. 618-627.

[19] C. Stroud, S. Wijesuriya, C. Hamilton, and M. Abramovici, "Built-In Self-Test of FPGA Interconnect," *Proc. IEEE Intn'l. Test Conf.*, 1998, pp. 404-411.

[20] Xilinx, Inc., *http://www.xilinx.com/products.*

Dr. Miron Abramovici is co-founder and CTO of DAFCA, a startup providing a reconfigurable infrastructure platform for Systems-on-Chip. Previously he was a Distinguished Member of Technical Staff at Bell Labs in Murray Hill, NJ. He is a Fellow of IEEE. He co-authored "Digital Systems Testing & Testable Design". He has over 70 publications and 17 issued patents. He has been a member of the editorial board of IEEE Design & Test of Computers and of Journal of Electronic Testing. Dr. Abramovici was Adjunct Professor of Computer Engineering at the Illinois Institute of Technology, Chicago, IL. He was principal investigator and project leader of a 3-year, DARPA-sponsored project on adaptive computing systems. He co-authored a paper included in the anthology "Twenty-Five Years of Electronic Design Automation."

Charles E. Stroud is a Professor in the Dept. of Electrical and Computer Engineering at the Auburn University. Previously he was a Professor in the Dept. of Electrical Engineering at Univ. of North Carolina at Charlotte and an Associate Professor in the Dept. of Electrical Engineering at Univ. of Kentucky. He worked for 15 years at Bell Labs in Naperville, IL, where he was a Distinguished Member of Technical Staff with primary areas of experience in digital circuit board and VLSI design, CAD tool development, and digital VLSI testing and diagnosis. He is a Senior Member of IEEE and author of "A Designer's Guide to Built-In Self-Test". He has over 100 publications and 12 issued patents.