BIST of Delay Faults in the Logic Architecture of Symmetrical FPGAs

Patrick Girard, Olivier Héron, Serge Pravossoudovitch, Michel Renovell

BIST of Delay Faults in the Logic Architecture of Symmetrical FPGAs

Patrick Girard Olivier Héron Serge Pravossoudovitch Michel Renovell
Laboratoire d'Informatique de Robotique et de Microélectronique de Montpellier
Université Montpellier II / CNRS (UMR 5506)
161, rue Ada - 34392 Montpellier Cedex 05, France
Email: <name>@lirmm.fr URL: www.lirmm.fr/~w3mic/

Abstract

In this paper, we propose a BIST scheme for exhaustive testing all delay faults in the logic architecture of symmetrical FPGAs. This scheme is applicable in a Manufacturing-Oriented Test (MOT) context. Our technique enables the detection of delay faults in the logic architecture and consists in chaining the logic cells in a specific way. The test of all the delay faults can be done with a reduced test sequence and does not require expensive ATE. To illustrate its feasibility, this BIST approach has been implemented in a VIRTEX FPGA from XILINX Inc.

1. Introduction

Field Programmable Gate Arrays (FPGAs) combine the flexibility of mask programmable gate arrays (MPGAs) with the convenience of field programmability. This technology has drastically reduced the cost of hardware, making hardware implementation economically feasible for applications previously restricted to software. Moreover, the convenience of field programmability associated to this technology has also brought some new design concepts, especially in the context of System-On Chips (SOCs). Actually, the use of reconfigurable logic gives to the system designers much greater flexibility to develop and to implement a design. Due to its reconfiguration property, more functionality can be addressed by the same piece of hardware. Consequently, FPGA technology is taking more and more significance for system designers and for the test community recently.

The FPGA testing can be viewed from two different ways: the manufacturer point of view or the user point of view. Techniques have been proposed to test either the whole FPGA structure before it is shipped to the user in a Manufacturing-Oriented Test (MOT) context [1, 2, 5, 10, 12, 15, 16, 17, 18, 19, 20] or only the used parts of the FPGA programmed for a user application in an Application-Oriented Test (AOT) context [11, 13, 14, 16]. The first proposed techniques can be used for static voltage testing [1, 10, 12, 15, 16, 17, 18, 19] and consider faults in the logic cells [1, 12, 16, 17, 18], in the routing architecture [1, 10, 17, 19], or in the configuration layer [16, 17]. Some of these techniques use a BIST architecture by configuring unused parts of the FPGA [2, 11, 18, 19].

As a result of the greater densities and more aggressive clock strategies, FPGAs have become more susceptible to delay faults. Unfortunately, testing delay faults in FPGAs is still an immature field and only few approaches have been proposed [2, 5, 11, 13, 14, 20]. In the remaining paragraph, we try to summarize the main points of recent published works. In [11, 13, 14], the objective is to detect delay faults in an AOT context. The approach developed in [13, 14] proposes to test each (critical) path of the design after customisation of the logic cells belonging to the tested path. For this purpose, the authors exploit the reconfiguration skill of the FPGA by replacing the user function defined in the Look-Up Tables (LUTs) of each logic cell of the tested path by a specific function. This is done to allow transition propagations on the tested path whatever the values on the side inputs and to preserve the possibilities of the input-output transition polarity of the original function. In [11], this process is applied in a BIST environment where a simple counter is used to control the LUT inputs and to generate the required transitions. In [2, 5, 20], the main objective is to detect delay faults in a MOT context. The main idea in [2] is to connect paths with the same number of LUTs and interconnections to a same transition generator. Every LUT implements an AND (or OR) function and, for each path, all the LUT inputs are connected to the same previous LUT output. In this way, each path propagates the same transition through LUTs that behave like buffers or “identity functions”. The detection of delay faults on the tested paths is obtained by comparing the propagation delays of the considered path to the other ones.

All these techniques use a path delay fault model in which a path is composed of programmed LUTs and interconnections. However, they only focus on faults located on the interconnections. Actually, the LUTs are handled as programmable black boxes and detecting delay

faults occurring within them is not considered explicitly. This is motivated by the fact that the interconnection delays represent the most significant part of the propagation delay in the application. In previous papers, we have shown the significant impact of spot defects on the timing of LUT internal paths and derived the conditions to detect all the delay faults within a single LUT [7, 8, 9]. In the present paper, we introduce a new BIST scheme applicable in a MOT context which allows an effective testing of delay faults within all the logic elements (LUTs) of a symmetrical SRAM-based FPGA in a short test time. This technique targets faults within LUTs and not faults on interconnections. So, this technique complements the solutions presented above that mainly target faults on interconnections.

The remainder of this paper is organized as follows. In section 2, some basics on the architecture of symmetrical FPGAs and backgrounds on delay fault testing in a single LUT are presented. In Section 3, we first present the test configuration for testing delay faults in the logic architecture of symmetrical FPGAs. Next, some delay considerations due to interconnection delays are discussed and the BIST architecture is described. Section 4 concludes the paper.

2. Basics and Backgrounds

2.1. Architecture of symmetrical FPGAs

A symmetrical FPGA architecture [3, 21] is composed of a two dimensional regular array of identical tiles, as shown in Figure 1. A tile is composed of a cluster-based logic block and surrounding routing channels.

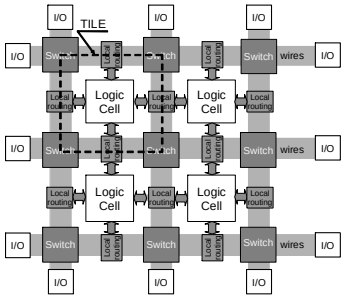

Figure 1. *Symmetrical FPGA architecture*

A logic cell is composed of *n*-input LUTs able to realize any combinational function of *n* inputs and D-type flip-flops. For reader convenience, we assume here a logic cell composed of one *n*-input Look-Up Table (LUT) and one D-type Flip-Flop (D), as shown in Figure 2. The demonstration can be easily extended to any type of configurable logic cells. The output of the logic cell can be either the registered or unregistered version of the LUT output according to the value of the output multiplexer (MuxOut). In this paper, we assume that the *n*-input LUT is a function generator which can implement any *n*-input logic function [21]. It is clear that this study can be easily extended to others configuration modes of the LUT. We assume that the LUT is composed of 2^n one-bit RAM cells, connected to a $2^n \rightarrow 1$ multiplexer, as shown in Figure 2 [3, 21]. The multiplexer can drive the value of any RAM cell towards the output. During the configuration mode, the LUT is programmed by setting every bit of the RAM cells at a value which is consistent with the truth table of the logic function. During the normal mode, the LUT can be viewed as an *n*-bit read-only memory. A pin of logic block can connect to some of the wiring segments in the channel adjacent to it via local routing wires. At every intersection of a horizontal channel and a vertical channel, there is a switch block which allows some of wires incident to it to be connected to others. Both switch and local routing blocks are simply a set of programmable switches.

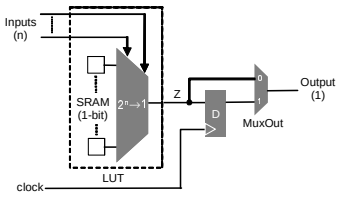

Figure 2. *Logic cell description*

The application configuration bits are loaded in the SRAM cells via dedicated pads [21]. The application realized by the circuit can be modified at any moment by reloading new configuration bits. This particularity of such programmable circuits can be used to make the test process easier [1, 5, 10, 11, 12, 14, 17, 18, 19, 20]. But, note that the reconfiguration time is several orders of magnitude greater than the pattern application time and then, the number of reconfiguration is a dominant factor determining the total test time.

2.2. Delay fault testing in an isolated logic cell

An *n*-input LUT is composed of 2^n internal paths connecting the 2^n SRAM cells to the LUT output *Z* [3, 4, 6]. The output Z displays the value stored in the SRAM cell R_i when the internal path P_i is activated i.e., when the input pattern $I_i = (E_{n-1}\ldots E_0)_i$ is applied on the LUT inputs $E_0\ldots E_{n-1}$. We have demonstrated in [8] that the complete delay fault testing of a LUT is achieved by activating all its internal paths, at least once time. This is done by applying a test sequence composed of 2^{n+1} input patterns, *n* being the number of LUT inputs. Depending on the function programmed in the LUT, we have also demonstrated in [8] that the test sequence can be reduced such that the minimal number of input patterns is equal to $2^n + 1$.

As an example, consider an *3*-input LUT programmed with the function $f(E_0 \ldots E_2) = \overline{E_0}$. This configuration is done by programming all the even SRAM cells R_0, R_2, R_4, R_6 at 1 and all the odd SRAM cells R_1, R_3, R_5, R_7 at 0. By using the procedure 2 in [8], we can determine a test sequence *TS* formed by the following test patterns: $\{I_0, I_1, I_2, I_3, I_4, I_5, I_6, I_7 \text{ and } I_0\}$. This test sequence is composed of $2^n + 1$ test patterns. Note that the opposite function $f(E_0 \ldots E_{n-1}) = E_0$ can lead to the same test sequence.

3. BIST in a Logic Cell Array

3.1. Test configuration scheme

In this section, we present a test configuration scheme that enables the test of all delay faults in the logic architecture of symmetrical FPGAs. The test with this scheme is done by using a reduced test sequence.

The solution we propose here consists in applying the same test sets to every LUT by chaining all the logic cells of the circuit in a special way. We use an approach similar to the one used to detect stuck-at faults in logic cells [17]. It consists in chaining alternatively a LUT and a D-type flip-flop available in the logic cells. Consider *k* *n*-input LUTs (or *k* logic cells) connected as follows: input E_0 of LUT *1* is the chain input a_0. The input E_0 of the LUT *i* ($i = 2 \ldots k$) is connected to the flip-flop output of the logic cell *i-1*. The output of LUT *i* ($i = 1 \ldots k$) is connected to the flip-flop input of logic cell *i*. The *n-1* inputs $E_1 \ldots E_{n-1}$ of all the LUTs share the *n-1* chain inputs, denoted as $a_1 \ldots a_{n-1}$. The output of the flip-flop *k* is the chain output, denoted as *s*. This configuration principle is illustrated in Figure 3.

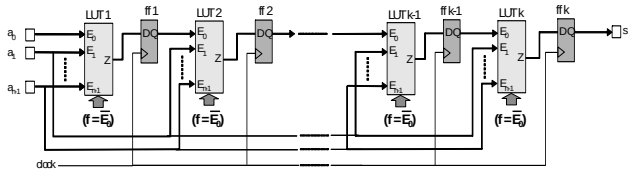

Figure 3. *Test configuration scheme*

Every LUT is programmed with the function $f(E_0 \ldots E_{n-1}) = \overline{E_0}$. In this way, each LUT can receive the same test pattern on its inputs (the test pattern is applied on the chain inputs a_i). In fact, the response of each LUT is transferred to the next LUT when it is captured by the flip-flop, i.e. at the next clock period. If a delay fault within a LUT is activated by a test pattern, the signal on its output is delayed. Thus, the flip-flop latches an erroneous value, producing a logic error on its output. This logic error is then serially propagated at each clock cycle to the next flip-flops until the output *s*. With such a configuration, testing delay faults in all the LUTs can be done by applying on the chain inputs a_i the same test sequence *TS* than the one defined in section 2.2 for a single LUT. At the end of the test sequence *TS*, note that *k* clock cycles are here required to completely unload the data stored in the flip-flop chain to the chain output *s*. The length of the test sequence depends only on the number of LUT inputs and not on the number of LUTs to be tested.

3.2. Implementation

Now, let us discuss the implementations in symmetrical FPGAs. Local and long wires can be used to connect a logic cell to another one and a pad to a pin of a logic cell in the FPGA. The synthesis tool generally optimises the circuit implementation by using local wires to connect adjacent cells. Thus, suppose that local wires are used to connect each flip-flop output to the input E_0 of the next logic cell, while long wires are used for propagating data from the input pads $a_1 \ldots a_{n-1}$ to the LUT inputs $E_1 \ldots E_{n-1}$. The signal propagation along the long wires takes more time than the one along the local wires. So, for each test pattern, we can consider that the transitions reach the input E_0 of each cell before the others inputs $E_1 \ldots E_{n-1}$. Moreover, this difference depends on the position of the LUT with respect to the input pads. It results that some delay faults normally detected by transitions on inputs $E_1 \ldots E_{n-1}$, may remain undetected if the delay induced by the fault is not large enough for producing an error in this configuration. To avoid this drawback, a solution is to modify the test sequence *TS* in order to avoid any timing dependency due to propagation delay along long wires.

The solution is to test the entire LUT by single bit transitions on input E_0. In the sequence *TS*, it exists some pattern pairs with a single bit transition on E_0, e.g. (I_0, I_1), (I_2, I_3), *etc*. Applying this sequence on the chain inputs allows activating half of the 2^n paths without any timing dependency. For testing the other half in similar conditions, the test sequence *TS* is modified as follows: the patterns $I_0, I_4 \ldots I_{2n-4}$ of pattern pairs (I_0, I_1), (I_4, I_5) … (I_{2n-4}, I_{2n-3}) are inserted at the end of them while the patterns $I_2, I_6 \ldots I_{2n-2}$ of pattern pairs (I_2, I_3), (I_6, I_7) … (I_{2n-2}, I_{2n-1}) are inserted at the beginning of them. The new test sequence is now composed of pattern tierces (I_0, I_1, I_0), (I_3, I_2, I_3), (I_4, I_5, I_4), $(I_7, I_6, I_7) \ldots$ $(I_{2n-1}, I_{2n-2}, I_{2n-1})$ and allows activating all the 2^n paths within the LUTs without any timing dependency. As a result, we can form a new test sequence TS_2 composed of 3.2^{n-1} test patterns as follows: $\{I_0, I_1, I_0, I_3, I_2, I_3, I_4, I_5, I_4, \ldots I_{2n-1}, I_{2n-2}, I_{2n-1}\}$

To obtain an efficient test for small delay faults as well, the test has to be performed with a specific clock scheme. Actually, the pattern pairs of TS_2 with a single bit transition, i.e. $(I_0, I_1, I_0), (I_3, I_2, I_3) \ldots (I_{2n-1}, I_{2n-2}, I_{2n-1})$ have to be applied at the higher frequency allowed by the local wires, while the others pattern pairs (pairs with multiple

bit transitions) can be applied at a lower frequency. The high frequency is determined by the propagation delay between two flip-flops. The low frequency is determined by the longest propagation delay between a chain input and the furthest flip-flop input. As a result, one should use a special clock signal formed by alternative fast and slow periods, as presented in Figure 5. In this figure, we show the test patterns (TS_2) applied on the chain inputs during each period. During the fast periods, the test of delay faults in each LUT is achieved in robust conditions i.e. the data on the LUT inputs $E_1 \ldots E_{n-1}$ are in a steady state. In Figure 5, this is done when the following patterns of the test sequence TS_2 are applied on the chain inputs: I_1, I_0, I_2, I_3, I_5, I_4 … I_{2^n-2}, I_{2^n-1}. During the slow periods, the remaining test patterns (from pattern pairs with multiple bit transitions) are set on the LUT inputs at a time depending on the mapping of each LUT. Such a pattern may appear as a test pattern and analyzing its response is not really useful to ensure the complete delay fault test of logic cells.

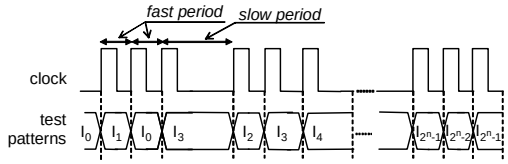

Figure 5. *Clock signal scheme*

3.3. BIST insertion

This section deals with the implementation of an accurate BIST architecture for testing delay faults in all the LUTs of a symmetrical FPGA. In addition of the test configuration scheme presented above, the objective is to implement a Test Pattern Generator (TPG) connected to the inputs a_i of the chain of LUTs on one side, and an Output Response Analyzer (ORA) connected to the chain output s on the other side. The BIST architecture does not need any area overhead in the circuit since the configurable logic cells are used to build the TPG and the ORA structures. After the test process, the BIST architecture can be removed from the circuit and the whole circuit can be used to implement a user-application. In addition, the test patterns are generated inside the circuit, avoiding any timing dependency due to long wires between some I/O pads and the input E_0 of the first LUT to be tested. Using the previous definitions, we now present the way the TPG is defined and the way the ORA is defined.

The TPG generates the test patterns of the test sequence TS_2 to be applied on the inputs a_i. Remember that this test sequence is composed of single bit transition pattern tierces (input E_0) and multiple bit transition pattern pairs. Its logic circuitry is shown in Figure 6. In this figure, we split the TPG in two finite state machines (FSM), as follows: FSM A generates the values to be applied on the input a_0 (or E_0 of LUT *1*) while FSM B generates the remaining values to be applied on inputs $a_1 \ldots a_{n-1}$ (or $E_1 \ldots E_{n-1}$ of LUTs). FSM A is a simple 2-modulo counter, using little logic. This is very important because it prevents the TPG determining the high frequency of the clock signal used to test the LUTs (Figure 5). FSM B is a 2^{n-1}-modulo counter in which the logic circuitry has a lower impact on the test. We define two clock schemes based on the one presented in Section 3.2 (Figure 5): fast test clock and slow clock. The fast test clock (high frequency) is applied to FSM A and to the chain of LUTs, while the slow clock (low frequency) controls FSM B. These clock signals can be separately generated but have to be synchronized between them, as shown in Figure 6. Signals fast test clock and slow clock are both connected to dedicated pads of the FPGA.

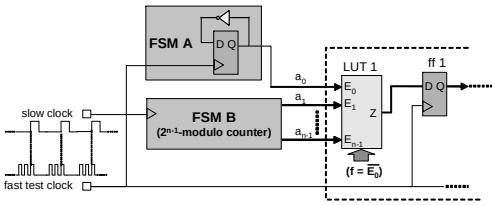

Figure 6. *TPG insertion scheme*

The ORA determines the correctness of the response provided by the output s at each clock cycle. For this purpose, a comparator compares each response to the expected one. The result is a pass or fail indication, at each clock cycle. To limit the amount of information, all the results are compressed by a compactor. This produces a single bit on an output, denoted as s_{ORA}, which is equal to 0 if no delay fault occurs in the logic cells or is equal to 1 otherwise. The ORA structure is presented in Figure 7. The comparator is composed of a XOR gate and a 2-modulo counter. It compares the output response with the fault-free value, produced by this counter. The result is latched by a flip-flop. At the next clock cycle, the value at the output of this flip-flop is compressed by a compactor, composed of a loop flip-flop with an OR gate. If a delay fault occurs in the logic architecture, the comparator provides a value 1 when the logic error arrives from the output s. This value is next compressed by the compactor that displays a steady state value 1 on the output s_{ORA}. All the flip-flops are initialized at 0 and are controlled by the fast test clock signal. Even if we have mentioned in the last section that the pattern pairs with multiple bit transitions may not be really useful for the testing, we assume here that the ORA verifies all the output responses.

Finally, the process we propose can use two different test sessions. In a first session, a given number of LUTs are used to build the TPG and the ORA structures and all the remaining LUTs, i.e. those not used

by the TPG and by the ORA, are tested. The LUT-under-test are involved in the test configuration scheme described in Section 3.1 that consists in building a chain of logic cells by chaining alternatively a LUT and a flip-flop. In contrast, it is quite obvious to imagine a split of this chain in several horizontal chains in order to decrease the time required to unload all the data in the flip-flops towards the ORA. This is done by building a given number of horizontal chains similar to the one given in Figure 3. The output *s* of every horizontal chain is connected to an ORA (Figure 7) on one side and its input a_0 is connected to a LSB state machine (Figure 6) on the other side. All the remaining chain inputs ($a_1 \ldots a_{n-1}$) share the same outputs of a single MSB state machine. An example of this test session is shown in a 10x4 symmetrical FPGA, in Figure 8. The black blocks represent the logic cells used by the TPG while the white blocks represent the ones involved in the ORA. The shaded area shows the part of the FPGA used by the MSB state machine. Dotted blocks represent the switch blocks that route the long wires from the MSB state machine to each logic cell under test. The local wires connect adjacent logic cells through the local routing boxes, not represented in this figure. The second configuration is required for testing delay faults in the logic cells used by the TPG and the ORA in the first session. The logic cells tested in the first test session are here involved to build the TPG and the ORA. The logic cells to be tested are connected in a way similar to the one of the first test session. Consequently, only two configurations and $3.2^{n-1} + k$ clock cycles by configuration (3.2^{n-1} test patterns), *k* being the number of flip-flops used, are needed to test all the delay faults in the logic architecture of symmetrical FPGAs, whatever their size.

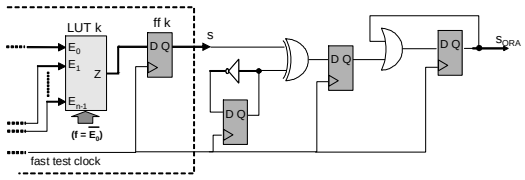

Figure 7. *ORA insertion scheme*

For validation, the BIST scheme has been implemented in a XCV50 VIRTEX FPGA from Xilinx Inc by using the XILINX synthesis tool (ISE). We have only considered one horizontal chain of logic cells, composed of 8 4-input LUTs and flip-flops ($k = 8$ *and* $n = 4$). The TPG is here composed of one LSB state machine and its MSB state machine is composed of an 8-modulo counter. All the LUTs are programmed with the function $f(E_0 \ldots E_{n-1}) = \overline{E_0}$. A timing analyzer available in the synthesis tool allows determining the frequency of the fast test clock signal and the slow clock signal. These are calculated after the mapping and routing of the BIST scheme in the FPGA. We have made timing simulations in two cases: fault-free case and faulty case. For this purpose, we have used the Mentor Graphics simulation tool (starter ModelSim XE II). Results from the simulations are shown in Figure 9.

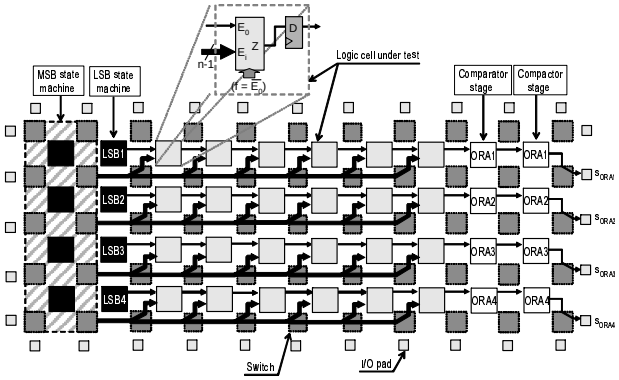

Figure 8. *BIST configuration example*

The figure should be interpreted as follows. We denote the clock signals as *'fastclk'* and *'slowclk'*, the asynchronous reset signal of flip-flops as *'raz'*, the output signals generated by the TPG as *'a0, a1, a2 and a3'*, the output signal of each flip-flop as *'q0, q1…q7 or s'* and the test result released by the ORA output as *'sora'*. In the left side, after the first initialization (raz = 1) the results show the fault-free mode of the BIST architecture. In this mode, it should be noted that the output *'sora'* is always at 0 and 10 clock cycles are here required at the end of the test sequence for the complete testing. To simulate a faulty mode, we have assumed faults in LUT *2*. So, we insert delay faults in LUT 2 by inverting the logic value stored in three of its SRAM cells: R_{12}, R_{13} and R_{15}. After the second initialization (right side), erroneous values appear on the output of LUT *2* when the following test patterns are applied on the LUT inputs: I_{13}, I_{12}, I_{13}, I_{15}, I_{14} *and* I_{15}. For reader convenience, we have marked the logic errors latched by the flip-flop ff_2 with arrows. When the first logic error reaches the ORA input (black arrow), the output *'sora'* is set at 1 two clock cycles later and remains in this state.

4. Conclusion

In this paper, we have proposed a BIST solution for testing delay faults in the logic architecture of symmetrical FPGAs, in a MOT context. The principle consists in chaining all the logic cells in an optimized way. The chain is formed by connecting each LUT output to the input E_0 of the next LUT through a flip-flop, available in every logic cell. A TPG is connected to the chain inputs and an ORA is connected to the chain output. The TPG generates the test patterns of the test sequence TS_2 and guarantees a delay fault testing in robust conditions. The ORA analyzes the response provided by the chain output and points out a pass or fail indication at

the end of the test process. On the benefits side, our BIST provides a test with large delay fault coverage within the logic architecture and reduced test sequence.

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", *IEEE Proc. of Int. Test Conf.*, pp 973-982, 1999.

[2] M. Abramovici and C. Stroud, "BIST-Based Delay-Fault Testing in FPGAs", *IEEE Proc. of Int. On-Line Testing Work.*, pp 131-134, 2002.

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

[4] R.A. Carberry, S.P. Young and T.J. Bauer, "FPGA LookUp Table with High Speed Read Decoder", *Xilinx Inc, San Jose*, United States Patent N° 6621296 B2, 2003.

[5] E. Chmelar, "FPGA Interconnect Delay Fault Testing", *IEEE Proc. of Int. Test Conf.*, pp 1239-1247, 2003.

[6] P. Chow, S.O. Seo, J. Rose, K. Chung, G. Páez-Monzón and I. Rahardja, "The Design of an SRAM-Based Field-Programmable Gate Array, Part II: Circuit Design and Layout", *IEEE Trans. on VLSI Systems*, Vol. 7 No. 3, pp 321-330, 1999.

[7] P. Girard, O. Héron, S. Pravossoudovitch and M. Renovell, "Defect Analysis for Delay-Fault BIST in FPGAs", *IEEE Proc. of Int. On-Line Testing Symp.*, pp 124-128, 2003.

[8] P. Girard, O. Héron, S. Pravossoudovitch and M. Renovell, "Requirements for Delay Testing of Look-Up Tables of SRAM-Based FPGAs", *IEEE Proc of European Test Work.*, pp 147-152, 2003.

[9] P. Girard, O. Héron, S. Pravossoudovitch and M. Renovell, "High-Quality TPG for Delay Faults in Look-Up Tables of FPGAs", *IEEE Work. on Electronic Design, Test and Application (DELTA'04)*, pp 83-88, 2004.

[10] I.G. Harris and R. Tessier, "Interconnect Testing in Cluster-Based FPGA Architectures", *Design Automation Conference*, pp. 49-54, 2000.

[11] I.G. Harris, P.R. Menon and R. Tessier, *"BIST-Based Delay Path Testing in FPGA Architectures"*, *Int. Test Conf.*, pp 364-369, 2001.

[12] W.K. Huang, F.J. Meyer, X.T. Chen and F. Lombardi, "Testing Configurable LUT-Based FPGAs", *IEEE Trans. on VLSI. Systems*, Vol 6, pp 276-283, 1998.

[13] A. Krasniewski, "Application-Dependent Testing of FPGA Delay Faults", *Proc. of EUROMICRO'99*, pp. 260-267, 1999.

[14] A. Krasniewski, "Exploiting Reconfigurability of Effective Detection of Delay Faults in LUT-Based FPGAs", *IEEE Proc. of Int. Conf. on Field Programmable Logic & Applications*, pp. 675-684, 2000.

[15] M. Renovell, J.M. Portal, J. Figueras and Y. Zorian, "SRAM-Based FPGAs: Testing the Embedded RAM Modules", *JETTA*, pp. 159-167, Vol 14, 1999.

[16] M. Renovell, J.M. Portal, P. Faure, J. Figueras and Y. Zorian, "Analyzing the Test Generation Problem for an Application-Oriented Test of FPGAs", *IEEE Proc. of European Test Work.*, pp. 157-162, 2000.

[17] M. Renovell and Y. Zorian, "Different Experiments in Test Generation for XILINX FPGAs", *IEEE Proc. of Int. Test Conf.*, pp. 854-861, 2000.

[18] C. Stroud, S. Konola, P. Chen and M. Abramovici, "Built-In Self Test of Logic Blocks in FPGAs", *IEEE Proc. of VLSI Test Symp.*, pp. 387-392, 1996.

[19] C. Stroud, E. Lee and M. Abramovici, "Built-In Self Test of FPGA Interconnect", *IEEE Proc. of Int. Test Conf.*, pp. 404-411, 1998.

[20] M. B. Tahoori, "Testing for Resistive Open Defects in FPGAs", *IEEE Proc. of Int. Conf. on Field Programmable Technology*, pp 332-335, 2002.

[21] Xilinx Inc, Virtex Series Product Specification (FPGAs), San Jose, USA, v2.6, 2002.

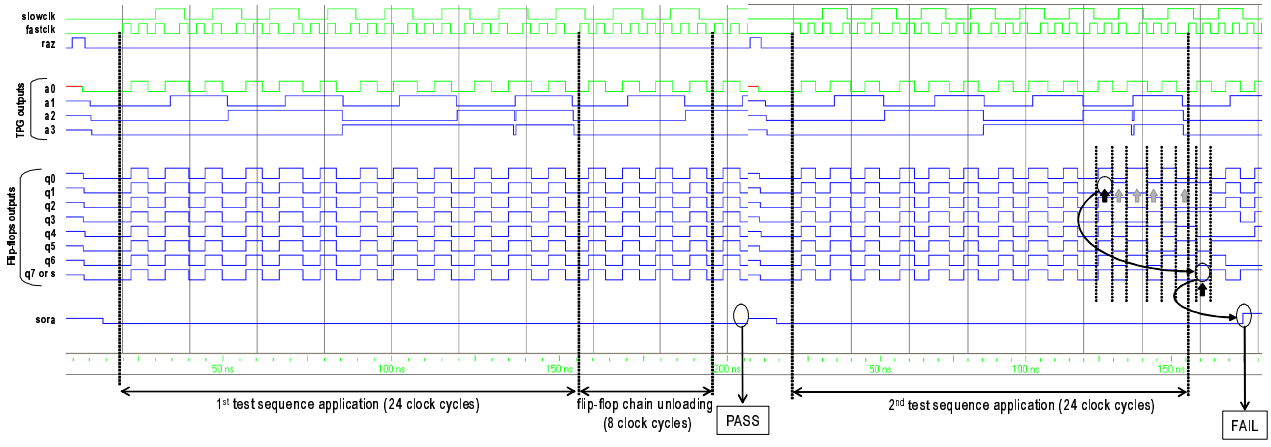

Figure 9. *BIST operations: fault-free mode (left side) and faulty mode (right side)*