Fine-grain Fault Diagnosis for FPGA Logic Blocks

Stavros Tzilis, Ioannis Sourdis, and Georgi N. Gaydadjiev
Computer Engineering
TU Delft, The Netherlands
emails: {S.Tzilis, I.Sourdis, G.N.Gaydadjiev}@tudelft.nl

Abstract—**In this paper we introduce a fine-grain fault diagnosis approach for reconfigurable logic blocks. As opposed to previous works, we propose to reuse rather than to discard defective blocks. We describe methods to analyze deeper a defective Xilinx Virtex2Pro slice and diagnose the fault, out of a set of 150, that causes the malfunction. The outcome of the fault diagnosis is subsequently used to characterize the defective slice functionality and then match it with a suitable design configuration. The proposed methods are implemented and prototyped in a Virtex2Pro-30. A single-phase fault diagnosis covers 95% of the faults and takes 170-390 nsec to test a single slice and 27-62 μsec for a frame of 160 slices. A two-phase approach uses reconfiguration and covers the entire set of faults requiring up to 8.5 μsec and 80 μsec for a single slice and an entire frame, respectively.**

I. Introduction

As feature size continues to shrink, transistors become less reliable while their count on a single chip rapidly increases. The combination of these two technology trends make the use of reconfigurable hardware ever more attractive. On one hand, the reconfigurable substrate is an excellent solution for defect tolerance; it is an array of identical substitutable units which can be configured and interconnected on demand to bypass defects. On the other hand, the area overhead of the reconfigurable hardware is becoming less significant due to the plethora of available on-chip resources.

A large number of related works advocate the use of FPGAs to provide fault tolerance, such as the works described in [1]–[7]. Most of them describe techniques for testing and locating defective units and subsequently substitute them with spare, correctly functioning ones. Naturally, a substitutable unit is a logic block (i.e. CLB, Slice or logic cell) or, at minimum, a LUT or Flip-flop. Such approaches are shown to be efficient for low defect-rates, however, they can be overly costly when defect-rates increase. In the coming nanoscale era, it is projected that a significant number of units will be defective at manufacture time and many more will degrade and fail over the expected lifetime of a chip [8], [9]. Consequently, even with the above granularity of substitutable units, significant part of the resources will be marked as defective and get discarded. A defective unit however, may still be useful; for example, a LUT with one defective SRAM-cell can still accommodate a large number of functions.

In our attempt to exploit the remaining capabilities of defective FPGA logic blocks, we investigate a finer-grain approach for defect-tolerance than previous works. We propose to perform (after testing) more detailed fault diagnosis and characterization of such logic blocks. In so doing, we identify a set of functions which can still be supported in a defective substitutable unit. A subsequent matching between these functions and the design configuration will potentially determine the way to successfully utilize the defective blocks. Our primary objective is to minimize the latency of the diagnosis and maximize the number of faults being diagnosed. As expected, similarly to FPGA testing, latency is dominated by the reconfiguration delay required between different diagnostic phases. Consequently, reducing the number of diagnostic phases will result is significant speedup.

In this paper we introduce two methods for fine-grain fault diagnosis of a FPGA Slice. More precisely, the following is performed:

- We propose a finer-grain fault diagnosis approach compared to related works to determine a single fault in a defective FPGA Slice out of a set of 150 possible faults. The fault diagnosis is subsequently used to characterize the capabilities of the slice.
- A two-phase method of fault diagnosis is described first, which requires a single reconfiguration and is able to cover 96% of the faults in a few μsec.
- A more sophisticated single-phase diagnosis method is subsequently proposed which covers 95% of the faults, does not need reconfiguration, and hence requires only a few tens of nsec. Adding a second phase to this method covers 100% of the faults.
- We designed and prototyped in a Virtex2Pro-30 device a tester for each of the above fault diagnosis methods and measure latency, diagnostic accuracy, and area cost.

The remainder of the paper is organized as follows: in Section II we provide some background on testing and fault diagnosis and discuss related works. In Sections III and IV we present our fault diagnostic methods and our characterization approach, respectively. In Section V, we evaluate our solutions and in Section VI we draw our conclusions.

II. Background

The two general approaches for FPGA defect tolerance are *sparing* and *matching* as described by A. DeHon in [9]. The first step for both of them is FPGA testing. Testing identifies whether each substitutable unit is defective, locates and returns a list of all defective units. Several approaches have been proposed for FPGA testing most of which are summarized in [1]. A recent work in FPGA testing is the method by Dutton and Stroud in [2] which tests a Virtex5 using 17 configurations to locate defects at the resolution of a LUT or flip-flop.

Sparing approaches use techniques to discard defective units and replace them with spare defect-free units. The defects are avoided either using defect-maps, which mark defective units, or creating perfect-components, discarding entire rows and columns of the reconfigurable array similarly to memories. Subsequently, spare, defect-free units are selected from a local cluster (local sparing) or from the entire device (global sparing) to substitute the defective ones. Cheatham et al. provide a detailed survey of works using sparing for FPGA fault tolerance in [3].

As opposed to sparing, matching analyzes further the units which according to the testing were reported as defective. This process is called fault diagnosis and identifies the exact fault that causes the unit to malfunction out of a set of faults. Such a set of possible faults is defined for a Circuit-under-Test (CUT) based on specific fault models. After fault diagnosis, the unit is characterized determining a list of functions which can be still supported correctly. These functions are then matched with possible configurations that may be mapped to the unit. Obviously, matching is a substantially more complex process than sparing, as it involves multiple comparisons between the possible functions and the available configurations, rather than a simple defect/defect-free decision required in sparing. Matching, however, is a more efficient process and therefore it is expected to (or alternatively can be forced to) have a small search space; that is, the number of different configurations checked before finding a match (or before failing to find one).

Next, we offer a brief overview of fault diagnosis approaches as they were categorized in [10] and [11]. Fault diagnosis may follow an effect-cause or a cause-effect methodology. The first one is based on the erroneous response of the CUT (the effect) to exclude one-by-one faults (causes) out of the set of possible faults until the fault which caused the response is found. On the contrary, the cause-effect diagnosis simulates a priori the faulty behavior (effect) of the CUT for each one of the faults (cause) in the predefined set of faults, and subsequently attempts to match the faulty behavior. A common cause-effect diagnosis approach uses diagnostic trees to analyze the responses of the CUT. A diagnostic tree is built based on the set of the suspected faults and the simulated response of the CUT for each fault. Each non-leaf node of the tree stores a test vector to be applied at the CUT and according to the CUT response the appropriate branch to the next level is selected. When a leaf is reached the diagnosis is completed and the fault causing the erroneous response is detected.

One of the most interesting works on FPGA testing, localization and diagnosis is the one by Abramovici, Stroud, Emmert et. al. in the framework of the Roving STARs paradigm [4]. This work, among other contributions, attempts to reuse FPGA faulty resources, having a small area of a few basic blocks roving around the whole FPGA for testing, diagnosis and fault tolerance purposes. These few logic blocks test each other through 15 different configuration phases. Abramovici et. al. identify defects at the granularity of a LUT and a flip-flop, considering as substitutable unit an ORCA-2C FPGA logic block. They discard defective resources (LUTs and/or flip-flops) rather than matching them to suitable functions. However, we consider that they implicitly perform some kind of coarse-grain characterization, as they go below the substitutable unit granularity and attempt to match the defect-free resources of a unit to the requirements of a neighboring configuration.

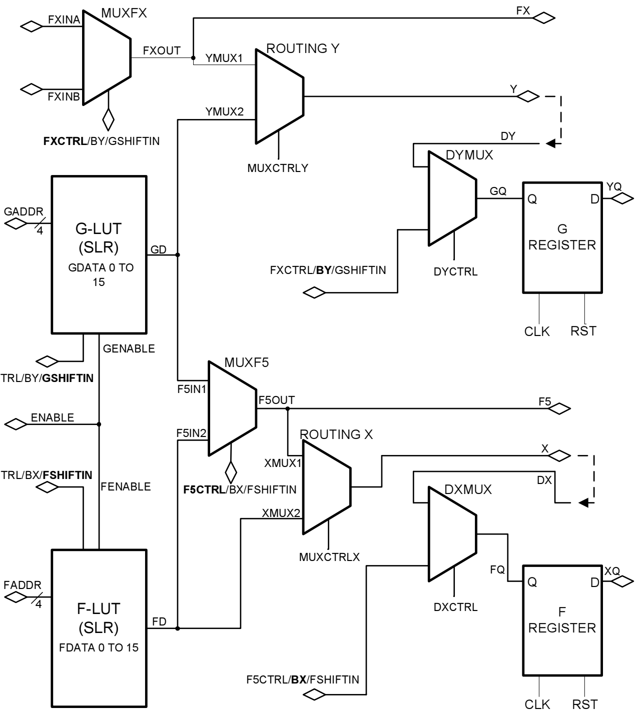
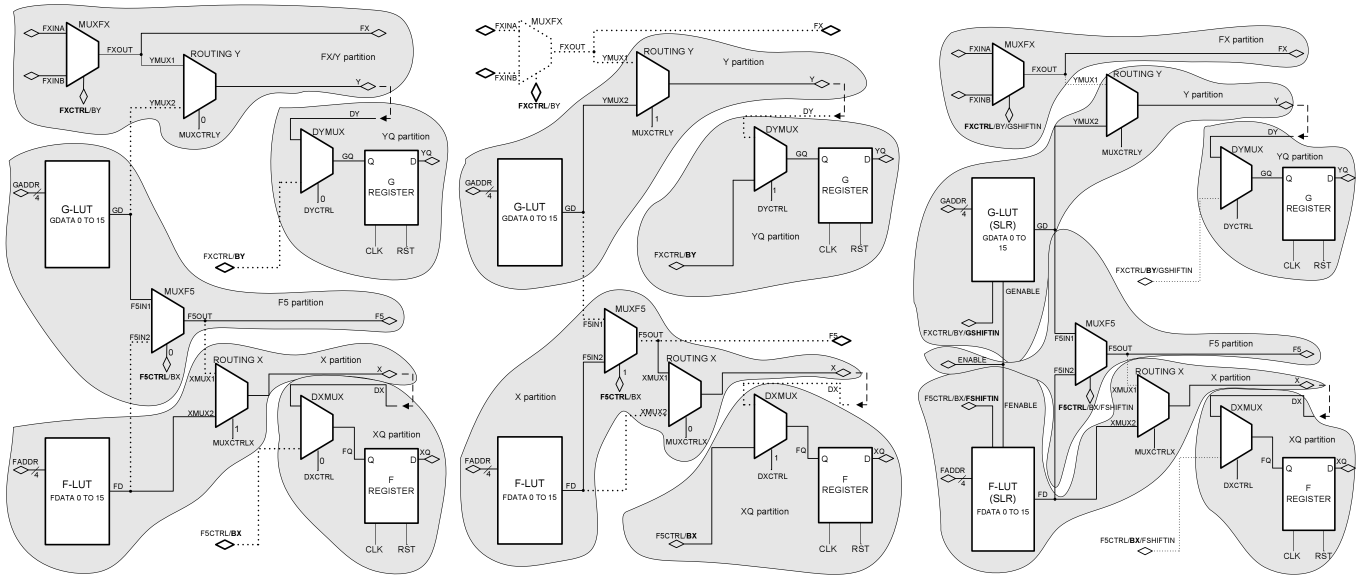


Fig. 1. The Virtex2-Pro slice diagram considered in our diagnosis methods.

In this work, we perform fault diagnosis in a higher resolution compared to previous works, and attempt to characterize and map configurations to defective resources. To exemplify, a multiplexer with a stuck-at-one fault at the select bit can be used when it fits a given configuration. As a logic block, we consider the Xilinx Virtex2-Pro Slice, illustrated in Figure 1, which consists of two LUTs, two flip-flops, and some additional multiplexers. For simplicity, the considered slice description omits the fast carry chain circuit and some arithmetic dedicated gates. Our fault model considers functional faults of two types, the first ones are stuck-at-zero and stuck-at-one faults for every wire depicted in Figure 1. The second type of faults is related to the storage elements of the Slice (LUTs and flip-flops), described based on the 3-tuple definition: <sensitizing-sequence/fault-effect/read-value> ($< S/F/R >$) by Al-Ars, van de Goor et. al. in [12]. According to [12], each bit of storage may experience 10 different faults, e.g., $< 0w1/0/- >$ describes the fault of writing 1 to a memory cell that stores a 0 and the state remains 0. Our fault model contains in total 150 (merged[1]) faults, each one affecting directly the functions that can be supported in the slice.

[1] E.g. all faults related to a single LUT-cell are merged to one entry in our fault model.



(a) Phase-1 of the FG method.

(b) Phase-2 of the FG method.

(c) SR method.

Fig. 2. The CUT partitioning for the phase-1 and 2 of the FG method, as well as for the SR method.

III. Fault Diagnosis Methods

Based on the above Slice description and its fault model definition, we have developed two methods for fault diagnosis to detect a single fault out of the 150 faults in the set. We follow a cause-effect methodology using diagnostic trees. In order to minimize the latency of the methods we aim at reducing the number of reconfiguration phases and the latency of each phase by reducing the depth of the diagnostic trees. To achieve this and to reduce the diagnosis complexity we divide the Circuit-under-Test (CUT) into partitions which can be checked independently. Then, a diagnostic tree is constructed for each partition, aiming at excluding as many faults possible at the earliest possible step. Obviously, the component that has higher diagnostic complexity is the LUT; there lies the primary difference between the two proposed methods.

In order to form our diagnostic approaches, we first observed how each fault of our model appears at the outputs of the CUT. The following useful conclusions were derived:

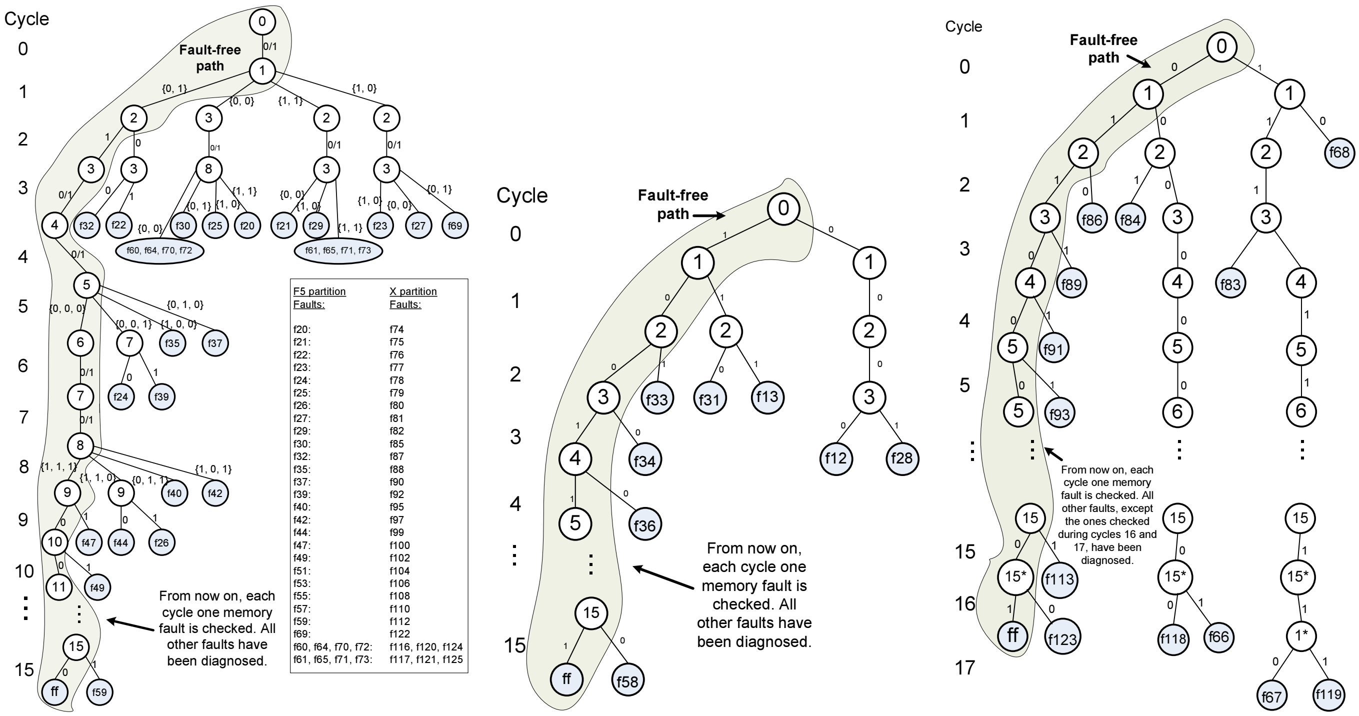
- A fault at a LUT-cell appears only when its content is read; this *singular effect* makes them easy to diagnose.
- When reading all LUT contents one by one, faults at an LUT address-line appear during *half the cycles (eight) of the process*. Whether or not the appearance manifests at an output, depends on the contents of the intended-to-be-read cell and the actually-being-read cell.
- Multiplexer faults have been reduced to faults of their selects. In essence, we are interested to check whether the multiplexer selects the correct input. To do so, we keep the inputs of each multiplexer complementary.
- Faults of wires force an output in their transitive fanout to be *fixed to a steady value*.
- Flip-flop faults are diagnosed on the cycle that the faulty transition (0→1, 1→0, 0→0, 1→1) occurs.
- Multiple faults may temporarily have the same faulty response. The diagnostic method will distinguish them in a subsequent step using the proper test vectors.

A. The Function-Generator Method

The main characteristic of this method is that the LUTs of the slice are configured as function generators (FG) that support any 4-to-1 boolean function. Given that we need to check every LUT-cell for faults in storing '0' and '1', it is obvious that this approach requires at least two phases and one reconfiguration. We explain below how the CUT is partitioned, how the LUT configuration is determined, and describe the diagnostic trees construction in the FG method.

As illustrated in Figure 1, the CUT has 6 outputs. Each fault on a component of the circuit can affect a subset of these outputs, which belong to the *transitive fanout* of the fault. For example, output Y is in the transitive fanout of the faults in YMUX2 wire, whereas output X is not. We follow the natural partitioning of the functionality among the parts of the slice and let each output be responsible to diagnose a well-defined subset of the faults. Partitioning is further affected by the position of some multiplexers. The selects of multiplexers ROUTING-Y, ROUTING-X, DY-MUX, and DX-MUX are driven by a configuration bit. Therefore, these selects are forced to be fixed during a single diagnostic phase. Essentially, in a single phase, the multiplexer output is fixed to a single input and at the same time the non-selected input is cut-off. By this cutting-off, the circuit is separated in independent parts. Figures 2(a) and 2(b) depict the partitioning of the CUT during the first and second phase of the FG method, respectively. It is worth noting, that the implementation of the above multiplexers has as a consequence the cut-off MUX-input to be untestable during the respective phase requiring the second phase to completely test it.

An interesting observation is that the inputs of some of the partitions are not primary inputs of the Slice and there-



F5 partition Faults:	X partition Faults:
f20:	f74
f21:	f75
f22:	f76
f23:	f77
f24:	f78
f25:	f79
f26:	f80
f27:	f81
f29:	f82
f30:	f85
f32:	f87
f35:	f88
f37:	f90
f39:	f92
f40:	f95
f42:	f97
f44:	f99
f47:	f100
f49:	f102
f51:	f104
f53:	f106
f55:	f108
f57:	f110
f59:	f112
f69:	f122
f60, f64, f70, f72:	f116, f120, f124
f61, f65, f71, f73:	f117, f121, f125

From now on, each cycle one memory fault is checked. All other faults have been diagnosed.

From now on, each cycle one memory fault is checked. All other faults have been diagnosed.

From now on, each cycle one memory fault is checked. All other faults, except the ones checked during cycles 16 and 17, have been diagnosed.

(a) F5 and X parts phase-1, FG method.

(b) Y part phase-2, FG method.

(c) X part phase-2, FG method.

Fig. 3. Diagnostic Trees of the FG method, phases 1 and 2, for the partitions that include the LUTs of the Slice under test.

fore cannot be explicitly controlled; we call such partitions "passive" as opposed to "active" ones. In order to diagnose passive partitions, we need to indirectly create the proper test-vectors through partitions that drive their inputs. Such passive partitions are the YQ and XQ of Figure 2(a). The inputs of the passive partitions are still exposed for read access to the tester, otherwise diagnostic tests would not be possible. For example, the YQ partition has one primary input (BY) and one internal (DY) which is however exposed since it is also the output Y of the slice.

The configuration of the LUT is based on a number of requirements posed by our fault models:

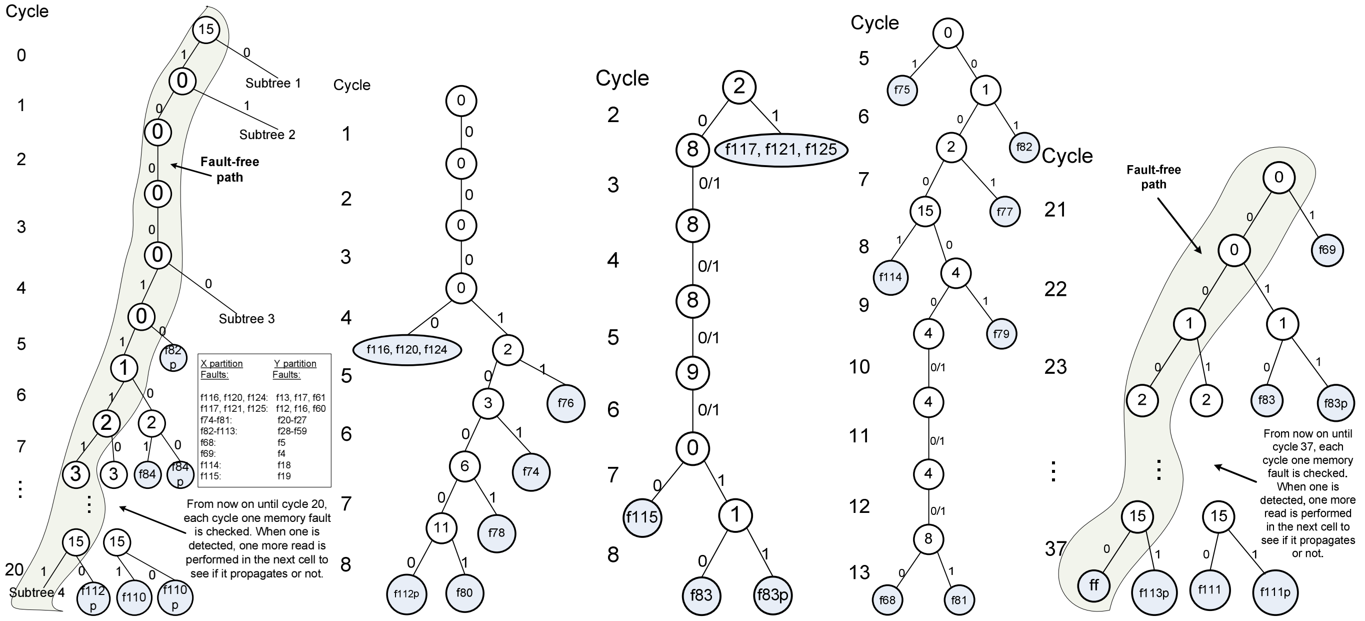
1) each LUT cell has to be read once during each phase in order to check whether it stores its value correctly.
2) in order to detect faults at the address lines of an LUT, at least two faulty reads need to be detected (out of the 8 that an address-line fault causes).
3) to test a multiplexer we need to keep their two inputs complementary; since the two LUTs provide input to MUXF5, we then configure the LUTs of the slice with complementary contents to test MUXF5.
4) in order to check the flip-flops at the output we need to feed them through the LUTs with the correct sequence of bits; that sequence is 10011 or 01100.

Based on the above constraints, we determine the LUT configurations and use a 4-bit counter to read the LUT contents one-by-one. An LUT configuration that satisfies the above constraints is (from bit 0 to 15) "0110001110000000" for G-LUT and its compliment for F-LUT, respectively. During the second phase, the two configurations are interchanged. Table I shows the addresses read and the responses of the fault free case for F-LUT, as well as the cases where there is a fault in an address line. When the fault is in a specific LUT-cell then the detection is trivial since the fault will appear in a single read out of the 15. In case of an address-line fault, the table shows in bold the observed faulty responses and underlined the faulty reads that go unnoticed. Each address-line fault requires at least two observed faulty responses to be detected. Finally, the multiplexers with configurable selects are configured as shown in the partitioning of the slice in Figure 2 to select one input and cut-off the remaining one.

TABLE I
LUT CONTENTS AND ACTUAL ADDRESSES READ IN THE FAULT-FREE CASE, AND IN CASE OF ADDRESS-LINE FAULTS. IN BOLD: WRONGLY READ ADDRESSES OBSERVED AT THE OUTPUT, UNDERLINED: WRONGLY READ ADDRESSES NOT OBSERVED DUE TO THE LUT CONTENTS.

LUT contents (0-15)	1	0	0	1	1	1	0	0	0	1	1	1	1	1	1	1
Fault Type	Actual Addresses Read															
Fault-free case	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
ADDR(0) s.a. 0	0	**0**	2	**2**	4	$\underline{4}$	6	$\underline{6}$	8	**8**	10	$\underline{10}$	12	$\underline{12}$	14	$\underline{14}$
ADDR(0) s.a. 1	**1**	1	**3**	3	$\underline{5}$	5	$\underline{7}$	7	**9**	9	$\underline{11}$	11	$\underline{13}$	13	$\underline{15}$	15
ADDR(1) s.a. 0	0	1	**0**	**1**	4	5	**4**	**5**	8	9	**8**	$\underline{9}$	12	13	$\underline{12}$	$\underline{13}$
ADDR(1) s.a. 1	**2**	**3**	2	3	**6**	**7**	6	7	**10**	$\underline{11}$	10	11	$\underline{14}$	$\underline{15}$	14	15
ADDR(2) s.a. 0	0	1	2	3	$\underline{0}$	**1**	$\underline{2}$	**3**	8	9	10	11	**8**	$\underline{9}$	$\underline{10}$	$\underline{11}$
ADDR(2) s.a. 1	$\underline{4}$	**5**	$\underline{6}$	**7**	4	5	6	7	**12**	$\underline{13}$	$\underline{14}$	$\underline{15}$	12	13	14	15
ADDR(3) s.a. 0	0	1	2	3	4	5	6	7	**0**	**1**	**2**	$\underline{3}$	$\underline{4}$	$\underline{5}$	**6**	**7**
ADDR(3) s.a. 1	**8**	**9**	**10**	$\underline{11}$	$\underline{12}$	$\underline{13}$	**14**	**15**	8	9	10	11	12	13	14	15

Having completed the formulation of the testing configuration (consisting of the LUT contents and the multiplexer selects), we construct the diagnostic trees for each partition and each phase of the diagnosis process. Partitioning the CUT allows as to have a Response Analyzer (RA) for each partition implementing the corresponding diagnostic tree, rather than having a common and substantially more complex RA for the



X partition Faults:	Y partition Faults:
f116, f120, f124:	f13, f17, f61
f117, f121, f125:	f12, f16, f60
f74-f81:	f20-f27
f82-f113:	f28-f59
f68:	f5
f69:	f4
f114:	f18
f115:	f19

From now on until cycle 20, each cycle one memory fault is checked. When one is detected, one more read is performed in the next cell to see if it propagates or not.

From now on until cycle 37, each cycle one memory fault is checked. When one is detected, one more read is performed in the next cell to see if it propagates or not.

(a) First part of the diagnostic tree. (b) Subtree 1. (c) Subtree 2. (d) Subtree 3. (e) Subtree 4.

Fig. 4. The diagnostic tree of the SR method for the partitions X and Y that include the LUTs of the Slice under test.

entire CUT. Figure 3 illustrates the largest diagnostic trees of our FG method; these are the trees for F5-partition in phase-1 (for X-partition is identical), and for Y and X partitions in phase-2, in Figures 3(a), 3(b), and 3(c), respectively.

To exemplify, we briefly describe the F5 diagnostic tree of phase-1. This tree is responsible for diagnosing all address-line faults of the G-LUT and also half of the LUT-cell faults. It is also responsible for only the stuck-at-0 (s.a.0) fault for the select of MUXF5 and for the faults in GD, F5IN1, F5OUT and F5 wires. To achieve that, a 4-bit counter addresses the G-LUT resulting in a read operation on every LUT entry.

After phase-1 of the method, some of the faults are completely diagnosed, others are partially resolved, while some remain to be checked in phase-2. In phase-1 all LUT address-line faults are checked as well as half of the LUT-cell faults, the MUXFX multiplexer, and faults in wires FXINA, FXINB, FXOUT and FX. The flip-flop and the remaining multiplexers are partially checked in phase-1 and completed in phase-2 together with the remaining LUT-cell and wire faults. Each phase takes 16 cycles, due to the LUT diagnostic tree, while the phase-2 requires two additional cycles for the ROUTING-X multiplexer. Consequently, the total latency of the FG method is 34 cycles plus the reconfiguration time of the slice.

B. The Shift-Register Method

Although our first method offers good diagnostic resolution, especially on the wires and multiplexers, its primary drawback is the need for reconfiguration between the two diagnostic phases; this essentially increases the diagnosis delay from a few hundreds of nsec to several μsec. In order to overcome this limitation we choose in our second diagnosis method to configure the LUTs as SRL16 shift registers. This enables us to shift in the LUTs new contents without requiring to reconfigure. This way, we are also able to diagnose memory faults in the LUT cells that previously were not checked. That is that the FG method was checking whether the LUT cells can store correctly their static contents; it could not check faults related to dynamic writes in the cells, e.g. $< 0w1/0/- >$, since the LUT contents could not change dynamically during a diagnosis phase. On the other hand, performing the diagnosis in a single phase has some drawbacks which affect the diagnostic accuracy[2] and resolution[3] of the Shift-Register (SR) method. More precisely, the absence of a second phase prevents us from checking most multiplexers[4] with both values of their select. We can overcome this limitation by adding a second phase to the SR method; this variation of the method is henceforth called SR2. However, the short latency of the SR method is substantially more important than the diagnostic accuracy and resolution. On one hand, being able to perform the diagnosis on the fly without reconfiguration makes SR method more attractive to be used online with the minimum overhead for a reconfigurable system. On the other hand, worsening the diagnostic accuracy and/or resolution may reduce the efficiency of the subsequent slice characterization, but does not affect its correct functionality. While a method for testing cannot afford to miss a defect, reducing the quality of fault diagnosis only affects the efficiency of the subsequent characterization and matching.

Our partitioning for the SR method is shown in Figure 2(c). As opposed to the FG method, the multiplexer MUXF5 is not used for the partitioning, but its select signal (F5CTRL)

[2] Diagnostic Accuracy is the percentage of all CUTs that are diagnosed correctly. When a CUT is known to be defective and all faults have equal probability to occur, diagnostic accuracy is the percentage of faults covered.

[3] Diagnostic Resolution is the average number of faults that cannot be distinguished with each other by a diagnostic method; the optimal diagnostic resolution is '1' corresponding to always detecting a single diagnosed fault.

[4] These are the multiplexers with configurable select.

TABLE II
THE LUT CONTENTS ON EVERY CYCLE OF THE SR METHOD.

Read Addr	15/0/0/0	0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
lut15	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0
lut14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0
lut13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0
lut12	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
lut11	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
lut10	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
lut9	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
lut8	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
lut7	0	0	0	0	0	0	0	0	1	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
lut6	0	0	0	0	0	0	0	1	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
lut5	0	0	0	0	0	0	1	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
lut4	0	0	0	0	0	1	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
lut3	0	0	0	0	1	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
lut2	0	0	0	1	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
lut1	0	0	1	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
lut0	0	1	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
cycle	0-3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37

is used as a variable input, in order to facilitate its testing. In order to test the flip-flops in a single phase, we need to choose whether to drive them directly or by the LUTs, since this cannot change dynamically during the single phase of the method. We choose to drive them by the LUT outputs in order to test the use of LUT and flip-flop in the same configuration. However, this choice adds a limitation to the possible functions mapped on the slice during matching; that is that a flip-flop needs to be always driven through the LUT regardless of it implementing or not a useful function. Finally, the G and F LUT outputs are propagated to the Y and X slice outputs, respectively. This way we check the basic functionality of realizing 4-input boolean functions in LUTs. Together with the complete MUXF5 test, this means that also the ability to accommodate 5-input boolean functions in the slice is checked.

The most important aspect of the SR method is the way in which it deals with memory (LUT-cell) faults. Since now we have access to the contents of the LUTs, we opted for developing a variation of a *march memory test*, described in [10], to cover the memory faults. The march test performs the following sequence of actions in a memory cell in order to detect all 10 memory fault types in our fault model: $\{\uparrow(w0, w1, w1, r1); \uparrow(w0, w0, r0)\}$. To apply the march test, we shift in the LUT the corresponding values (01100), and perform the appropriate reads in each LUT cell in order to check it. Before the march test, we initialize properly the contents of the LUTs, and spend four cycles to check the enable and shift-in pins of the shift register as well as to partially check for LUT address-line faults. Finally, the values shifted in the LUT after the ones of the march test are not important for the march test. We make use of these values to complete the diagnosis of the LUT address-line faults. The sequence of bits shifted in the LUT as well as its contents in each cycle are illustrated in Table II; the two LUTs are checked in an identical way with complimentary contents.

After partitioning the slice and defining the diagnosis approach for the LUTs, we construct the diagnostic tree of each partition similarly to the first method. Figure 4 depicts the most complex tree used for the response analysis of the X-partition which includes the F-LUT (the tree for the Y-partition is identical). The fault-free scenario is the longest one requiring 38 cycles. The tree has three subtrees followed in the case if address-line faults and faults at the enable and shift-in pins of the LUT. The X and Y diagnostic trees determine the overall latency of the method, which is 38 cycles.

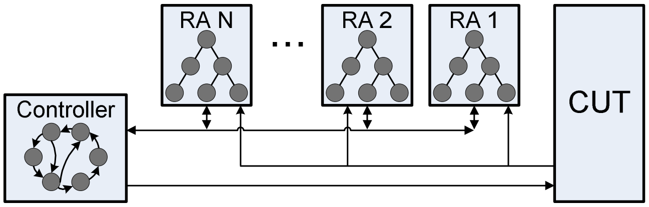


Fig. 5. Tester Block Diagram

C. Tester

Each fault diagnostic method is performed by a tester implemented inside the FPGA. This allows to perform the diagnosis online and support a BIST approach. Figure 5 shows the block diagram of the tester which consists of a centralized control unit and a number of Response Analyzers (RA). Each RA implements the diagnostic tree corresponding to a CUT partition. The RAs provide feedback to the control unit which implements the top-level Finite State Machine of the tester, feeds the CUT with the proper test-vectors, and reports the detected faults. The basic advantage of this approach is that it can support an adaptive diagnostic strategy, by identifying the set of suspected faults after each fragment of the CUT response and by making a decision as soon as possible. In some cases, it is even profitable to modify the upcoming test vectors in order to target the suspected faults specifically and reach the conclusion even sooner. This means that the RA should give some feedback to the controller, in order to determine the subsequent steps.

IV. CHARACTERIZATION

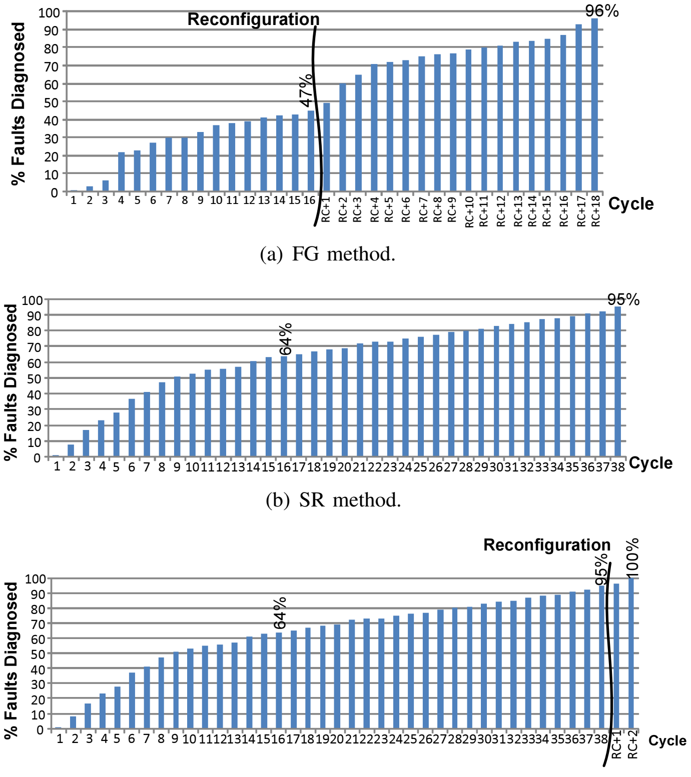
The results of the fault diagnosis process provide a list of faults for the slice under test. This information is then used for the characterization step to indicate the functionality supported by the defective slice. Previous works determine the subset of functions supported by the defective block based on the reported faults. In this work, we opted for a slightly different approach. We defined a priori a set of modes which would allow the defective device to gracefully degrade based on the fault diagnosis. Each fault maps to one of the above modes, while each mode constraints the configuration of the block to a predefined subset of supported functions. This way matching can be relatively simpler, by inspecting a few bits of the intended configuration bit stream. All modes of degradation and the faults that result to each of them according to the FG, SR and SR2 methods are listed in Table IV. Finally, the characterization is independent from the diagnosis; consequently, it is simple to choose another set of modes of degradation, or another manner of characterization, and still use the same results of our diagnostic methods.

TABLE III
MODES OF GRACEFUL DEGRADATION

mode	FG faults	SR faults	SR2 faults	Description
M0.0	-	-	Fault-free	The CUT is fault-free.
M0.1	Fault-free	-	-	The CUT is fault-free. Limitations: the LUTs are not configured as shift-registers.
M0.2	-	Fault-free	-	The CUT is fault-free. Limitations: a flip-flop is driven through the respective LUT, some MUXs can be used only with the tested configuration.
M1	f0-7	f0-9	f0-7	MUXFX not functional, output FX not usable. Signal YMUX1 not usable, hence ROUTING Y multiplexer usable only with MUXCTRLY = '1'.
M2	f8-9	-	f8-9	Output FX not usable.
M3	f10-11, f15	-	f10-11, f15	ROUTING Y multiplexer usable only with MUXCTRLY = '1'.
M4	f12-14	-	f12-14	ROUTING Y multiplexer usable only with MUXCTRLY = '0'.
M5	f16-17	f12-13, f16-17	f16-17	Output Y not usable. Y flip-flop only usable directly through input BY (DYCTRL = '1').
M6	GLUT 8 addr faults			G-LUT only usable as a 3-variable FG, excluding the faulty address line. M6 has 8 different modes depending on the address line that is faulty and whether it is s.a.0 or s.a.1 fault.
M7	GLUT 32 mem faults			G-LUT entry needs to be configured to the faulty value. M7 has 32 different modes depending on the faulty memory cell and s.a value of the cell.
M8	f60-61			G-LUT not usable. MUXCTRLY has to be configured to '0'.
M9	f64-71	f64-73	f64-71	MUXF5 not usable. Output F5 not usable. Output X can only be driven by the F-LUT (MUXCTRLX = '1').
M10	f72-73	-	f72-73	Output F5 not usable.
M11	FLUT 8 addr faults			F-LUT only usable as a 3-variable FG, excluding the faulty address line. M6 has 8 different modes depending on the address line that is faulty and whether it is s.a.0 or s.a.1 fault.
M12	FLUT 32 mem faults			F-LUT entry needs to be configured to the faulty value. M7 has 32 different modes depending on the faulty memory cell and s.a value of the cell.
M13	f116-117			F-LUT not usable. Outputs F5 and X not usable. X flip-flop only usable directly (DXCTRL = '1').
M14	f118-119, f123	-	f118-119, f123	ROUTING MUX X can only propagate the F-LUT output to output X (MUXCTRLX = '1').
M15	f120-122	-	f120-122	ROUTING MUX X can only propagate MUXF5 output to output X (MUXCTRLX = '0').
M16	f124-125	f120-121, f124-125	f124-125	Output X not usable. X flip-flop only usable directly, through input BX (DXCTRL = '1').
M17	f126-127, f129	f129	f126-127, f129	Y flip-flop only usable directly, through input BY (DYCTRL = '1').
M18	f128	-	f128	Y flip-flop only usable through logic output Y (DYCTRL = '0').
M19	f130-137	f126-127, f130-137	f130-137	Y flip-flop not usable. YQ output not usable.
M20	f138-139, f141	f141	f138-139, f141	X flip-flop only usable directly, through input BX (DXCTRL = '1').
M21	f140	-	f140	X flip-flop only usable through logic output X (DXCTRL = '0').
M22	f142-149	f138-139, f142-149	f142-149	X flip-flop not usable. XQ output not usable.
M23	-	f18-19	f18-19	G-LUT unable to be used as shift register.
M24	-	f114-115	f114-115	F-LUT unable to be used as shift register.
M25	-	f62-63	f62-63	Both LUTs unable to be used as shift registers.
M26	-	-	-	Slice not usable at all.

V. EXPERIMENTAL RESULTS

We have implemented the FG and the SR methods, as well as a third method of the Shift-Register approach with a second phase to cover the remaining multiplexer faults (SR2), using a Xilinx Virtex2Pro device. After simulating the faults in our fault model and constructing the diagnostic trees for each partition of the methods, we implemented the testers for the three methods. The testers were prototyped in a Virtex2Pro-30 and performed the fault diagnostic tests in real FPGA slices. In this Section, we report the percentage of covered faults, the diagnostic latency, as well as the diagnostic accuracy and resolution of each case. We further present the area cost and operating frequency of the three testers. Finally, we measure latency when testing an entire FPGA frame; that is the minimum Virtex2-Pro reconfigurable block that spans the entire device height and a fraction of one column and consists of 160 slices.



(a) FG method.

(b) SR method.

(c) SR2 method.

Fig. 6. Cumulative percentage of diagnosed faults on every cycle.

Figure 6 depicts the cumulative percentage of faults diagnosed by each method at every cycle; obviously, the fault-free cause is only detected at the last cycle of the process. The FG method, in Figure 6(a), detects 47% of the faults in phase-1 and goes up to 96% at the end of phase-2, since it is not able to resolve faults related to the shift-register mode of the LUTs. As depicted in Figure 6(b), the SR method detects in a single phase 95% of the faults in our fault model, covering only partially the multiplexer faults. After adding a second phase, the SR2 method is able to cover the remaining 5% of the faults using two additional cycles.

Table IV shows the implementation results of the testers and summarizes the characteristics of the three methods. In all cases the area cost of the tester is 250-280 Slices while the operating frequency is above 100 MHz. The latency of each method depends on the specific fault diagnosed. The process is terminated when a fault is detected, otherwise it continues until the last step which determines the CUT fault-free. We consider equal probability for each fault in the set and measure latency for two cases: (i) when we know that the CUT is always defective, and (ii) when the probability for the CUT to be faulty is $P_{faulty} = 1/4$; that is e.g. when the testing and locating process detects faulty CLB rather than slices. In the

TABLE IV
IMPLEMENTATION RESULTS

Measure	FG method	SR method	SR2 method
Tester Area	250 slices	272 slices	280 slices
Tester Frequency	110Mhz	103Mhz	103Mhz
Latency (faulty CUT)	4,769 ns	170 ns	607 ns
Latency ($P_{faulty} = 1/4$)	7,869 ns	348 ns	6,911 ns
Diagnostic Accuracy (faulty CUT)	0.96	0.947	1
Diagnostic Accuracy ($P_{faulty} = 1/4$)	0.99	0.987	1
Diagnostic Resolution	1.25	1.37	1.24

fist case, the latency of the FG method is substantially greater than that of the SR and SR2 due to the reconfiguration[5]; that is 28 and 8× slower, respectively. The SR2 method has only 3.5× more latency than the SR, since only 5% of the faults require a second phase. In the second case latency doubles for the FG and SR method, while for the SR2 increases more than 11× since the higher probability to have a fault-free CUT increases the probability of reconfiguration. Furthermore, Table IV offers the diagnostic accuracy and resolution of the methods. The accuracy of the SR2 is 1 since it covers all the faults, while for the other two methods the accuracy increases for lower P_{faulty}. The diagnostic resolution of the methods is 1.24-1.37 as some faults are not distinguished.

We next evaluate in more detail the latency of the three fault diagnostic approaches. Latency in FG an SR2 methods is dominated by the reconfiguration delay mainly because in Virtex2Pro-30 we cannot reconfigure a single slice alone, but only large frames of 160 slices. We consequently measure the latency of the three methods when testing a single slice as well as when testing sequentially all slices in an entire frame and share the reconfiguration penalty. We further vary the probability of the CUT to be defective from $P_{faulty} = 1$ to $1/16$ depending on the accuracy of the (former) localization step. Figure 7 shows the latency for the first case; as expected the SR method is faster requiring only 170 ns and up to 393 ns for lower P_{faulty}. SR2 and FG methods require up to 8.6 μsec due to the reconfiguration overhead. In the case of diagnosing an entire frame, the results are more comparable between the three methods (Figure 8). SR is still faster requiring 27-63 μsec, but for low P_{faulty} FG is as fast; The SR2 that covers 100% of the faults requires 37-80 μsec.

VI. CONCLUSIONS

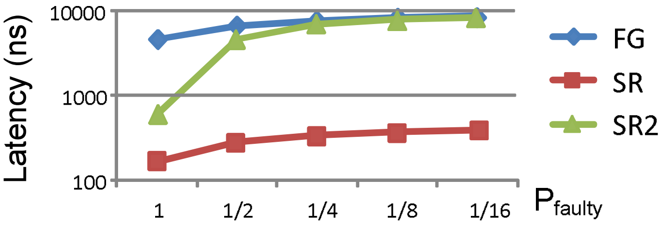


Fig. 7. Comparative Latency graph for the FG, SR, and SR2 methods.

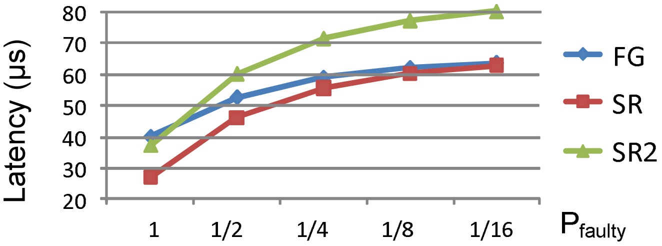


Fig. 8. Comparative Latency graph for the FG, SR, and SR2 methods.

As technology scaling increases defect-rates, characterizing and reusing, rather than discarding, defective blocks is a more efficient solution. In this work, we proposed a finer-grain FPGA defect tolerance approach compared to previous works. We introduced new fault diagnostic methods for Virtex2Pro logic blocks to determine a single fault, out of a set of 150 faults, in a slice. The detected fault is subsequently used for the characterization step which defines the possible functions still supported by the defective slice. We prototyped and evaluated our proposal using a Xilinx Virtex2Pro-30 and discussed the tradeoff between diagnostic accuracy and latency. Our FG method configures the LUTs as function generators, and requires two phases to detect 96% of the faults taking several μseconds. The diagnostic latency of the more advanced single-phase Shift-Register (SR) method, is a few tens of nanoseconds and detects 95% of the faults. In order to cover 100% of the faults a two-phase SR2 approach is introduced requiring 8.5 μsec to analyze a single slice and and 80 μsec for an entire frame of 160 slices.

REFERENCES

[1] A. Doumar and H. Ito, "Detecting, diagnosing, and tolerating faults in SRAM-based field programmable gate arrays: A survey," *IEEE Trans. VLSI Syst*, vol. 11, no. 3, pp. 386–405, 2003.

[2] B. Dutton and C. Stroud, "Built-in self-test of configurable logic blocks in virtex-5 fpgas," in *System Theory, 2009. SSST 2009. 41st Southeastern Symposium on*, 15-17 2009, pp. 230 –234.

[3] J. A. Cheatham, J. M. Emmert, and S. Baumgart, "A survey of fault tolerant methodologies for fpgas," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 11, no. 2, pp. 501–533, 2006.

[4] J. M. Emmert, C. E. Stroud, and M. Abramovici, "Online fault tolerance for fpga logic blocks," *IEEE Trans. VLSI Syst.*, vol. 15, no. 2, pp. 216–226, 2007.

[5] M. Niamat, K. Attravanam, and M. Alam, "Testing fpgas using jbits rtp cores," in *48th Midwest Symp. on Circuits and Systems*, 7-10 2005, pp. 1131 –1134 Vol. 2.

[6] P. Sundararajan and S. A. Guccione, "Run-time defect tolerance using jbits," in *ACM/SIGDA Int. Symp. on FPGA*. New York, NY, USA: ACM, 2001, pp. 193–198.

[7] S.-J. Wang and T.-M. Tsai, "Test and diagnosis of fault logic blocks in fpgas," in *IEEE/ACM Int. Conf. on Computer-aided design (ICCAD)*. Washington, DC, USA: IEEE Computer Society, 1997, pp. 722–727.

[8] S. Borkar, "Designing reliable systems from unreliable components: the challenges of transistor variability and degradation," *Micro, IEEE*, vol. 25, no. 6, pp. 10 – 16, nov.-dec. 2005.

[9] S. Hauck and A. DeHon, *Reconfigurable Computing: The Theory and Practice of FPGA-Based Computation*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2007, ch. 37: Defect and Fault Tolerance, dy A. DeHon, pp. 829–852.

[10] S. G. Niraj Jha, *Testing of Digital Systems*. The Press Syndicate of the University of Cambridge, 2003.

[11] V. D. A. Michael L. Bushnell, *Essentials of Electronic Testing*. Springer Science + Business Media, LLC, 2000.

[12] A. J. van de Goor and Z. Al-Ars, "Functional memory faults: A formal notation and a taxonomy," in *VTS*, 2000, pp. 281–290.

[13] P. N. Sedcole, B. Blodget, T. Becker, J. Anderson, and P. Lysaght, "Modular partial reconfiguration in virtex fpgas," in *Int. Conf. on Field Programmable Logic and Applications (FPL)*, 2005, pp. 211–216.

[5] We consider that the reconfiguration of a single frame is 8.547 μsec, as accurately measured in [13], considering that the configuration is preloaded in the ICAP.