A Novel Approach to FPGA-Based Hardware Fault Modeling and Simulation

Abílio Parreira, J. P. Teixeira and Marcelino Santos
IST / INESC-ID, R. Alves Redol, 9, 1000-029 Lisboa, Portugal
marcelino.santos@inesc.pt

Abstract: *The purpose of this work is to present a Hardware Fault Simulation (HFS) methodology and tool (f²s), using partial reconfiguration, suitable for efficient fault modeling and simulation in FPGAs. The methodology is particularly useful for BIST effectiveness evaluation, and for applications in which multiple fault injection is mandatory, such as in safety-critical applications. A novel (CSA – Combination Stuck-At) fault model is proposed, which leads to better test quality estimates than the classic (LSA – Line Stuck-At) model at the LUTs terminals. Fault injection is performed using only local reconfiguration with small binary files. The efficiency of Software and FPGA-based HFS, with and without partial reconfiguration, are compared using ISCAS'89 sequential benchmarks, showing the usefulness of the proposed methodology.*

1 – Introduction

Design for Testability (DfT) techniques are mandatory for cost-effective product development. Full scan based DfT techniques are popular for sequential digital modules. However, it has limitations, due to the impact on device performance, test overhead, test application time and no at-speed testing, which is crucial to catch dynamic faults [1]. Hence, partial scan or self-test strategies have been used, requiring testability analysis of sequential blocks.

In the design environment, test quality validation is performed through Fault Simulation (FS). For complex devices, FS may be a very costly process, especially for sequential digital modules. In fact, circuit complexity, test pattern length and fault list size may lead to a large computer effort. Although many efficient algorithms have been proposed (see, e.g., [2] [3]), fault simulation in complex circuits is still a very time-consuming task and can significantly lengthen the time-to-market.

FS can be carried out implementing the process in software, or in hardware [1]. The easiness in developing software tools for FS (taking advantage of the easiness of software programmability) made this implementation widespread. However, the availability of very complex *hardware* programmable devices, namely FPGAs (Field Programmable Gate Arrays), opened the way to make HFS (*Hardware Fault Simulation*) an attractive solution,

at least for a subset of practical situations. In fact, Built-In Self-Test (BIST) effectiveness may require computational heavy fault simulation effort, due to long test sessions and if several modules must be present in order to evaluate aliasing – e.g., functional BIST in Systems on a Chip (SoCs) with data compression cores being used as part of the signature compression logic. Another fault simulation task not efficiently performed by software fault simulation tools is *multiple* fault injection, due to the enormous number of possible combinations. However, multiple fault simulation may become mandatory, namely in the certification of safety critical applications [4].

Different HFS approaches, using FPGAs, have been proposed to overcome SFS (Software Fault Simulation) difficulties with sequential circuits. *Dynamic* fault injection was proposed [5][6][7][8], by adding different amounts of extra hardware, allowing the injection of different faults without reconfiguring the FPGA. The additional hardware proposed for implementing the dynamic fault injection uses a shift-register (SR) with one bit for each fault to inject. Each fault is injected when the corresponding position in the SR has a logic "1". In the initialization, only the first position of this SR is set to "1". This "1" is then shifted, activating one fault at a time. However, this technique does not optimize the HFS speed for all the situations [9]. Moreover, the added hardware increases with the number of faults to inject, thus limiting the size of the circuits to simulate. [6] also introduces some parallelism in HFS by injecting at the same time the faults identified as independent. This type of parallelism allowed a reported speedup of 1.36 over serial fault simulation. [10] proposed a new approach that included a backward propagation network to allow critical path tracing [11]. This information allows multiple faults to be simulated for each test vector. However, in [10] only combinational circuits are analyzed.

In order to simulate large circuits no extra logic must be added for fault injection purposes. Taking this approach, [12] proposed a serial fault simulation technique that required only partial reconfiguration of a logic emulation board, and concluded that HFS should be two orders of magnitude faster than SFS for designs with 100.000 gates. More recently, hardware fault injection approaches were proposed in [13-15] using partial FPGA reconfiguration through a Jbits [16] interface. JBits software development kit (SDK) allows partial reconfiguration of FPGA suitable for high efficient fault injection on Look-Up-Tables (LUTs) [13] [14] or erroneous register values [15]. Partial FPGA reconfiguration allows high efficiency in hardware fault injection. However, previous work does not report any results on partial FPGA reconfiguration for fault injection, only some predictions are made. Moreover, JBits SDK requires Java 1.2.2 [17] and the XHWIF [18] hardware interface.

The purpose of this paper is to present a new HFS methodology, using partial reconfiguration, and a novel, highly efficient hardware fault simulation tool, **f²s**, that injects faults using small bit files. These bit files are obtained from the full configuration by means of direct bit stream manipulation, thus not requiring any additional software. Additionally, the new HFS tool does not force the use of any specific hardware interface with the FPGA. As the technology mapping in a FPGA differs from the one in an ASIC, a new fault model is proposed to ascertain the quality of the test pattern.

The paper is organized as follows. In section 2 the LUT extraction from the bit stream is explained, the new fault model and a companion fault collapsing technique are proposed. In section 3 the new fault simulation tool, f²s, is presented. Section 4 presents results, which compare software fault simulation and FPGA-based HFS, with and without partial reconfiguration, using ISCAS'89 benchmark circuits [19]. Finally, section 5 summarizes the conclusions of the work.

2 – Fault Models and FPGA HFS

2.1 – Virtex FPGA LUT extraction

Virtex FPGAs [20] are composed of an array of Configurable Logic Blocks (CLBs) surrounded by a ring of I/O blocks. The CLBs are the building blocks for implementing customizable logic. Each CLB contains two slices. Each slice contains two 4 input Look-Up-Tables (LUTs), 2 flip-flops and some carry logic. For the fault models proposed in the next section, only LUT extraction is required. Each LUT can be used to implement a function of 0, 1, 2, 3 or 4 inputs. It is important to identify the number of inputs relevant for each used LUT in order to include the corresponding faults in the fault list. Since each LUT has 4 inputs, the LUT contents consist on 16 bits, one for each combination of the inputs. Let y_{abcd} be the output value for the combination of input values *abcd*. Each LUT information in the bit stream can be viewed as a 16 bit vector, (y_{0000}, y_{0001}, y_{0010}, …, y_{1111}), that corresponds to the LUT output values for the possible combinations of inputs i_3, i_2, i_1 and i_0. When fixing a value of one input, a 8 bit vector is obtained. For instance, if we consider only the possible combinations for i_3=0 the 8 bit vector is: (y_{0000}, y_{0001}, y_{0010}, y_{0011}, y_{0100}, y_{0101}, y_{0110}, y_{0111}).

The only circuit information required for the HFS is the bit file. After retrieving the information for each LUT [21] from the bit file, the relevance of each input i_x (x=0,1,2 and 3) is evaluated comparing the 8 bit vectors corresponding to i_x=0 and i_x=1. If these two vectors are identical the input i_x is not active.

2.2 – Line Stuck-At (LSA) Fault Injection and Collapsing

The most popular fault model for digital IC fault simulation is the single Line-Stuck-At (LSA) fault model. HFS in Virtex FPGAs can easily inject LSA faults in outputs and inputs of LUTs. After knowing the number of active inputs in each LUT, it is possible to create the LSA fault list, consisting on two LSA faults for each input plus two LSA faults for the outputs. Thus, the total number of faults is 2, 4, 6, 8 and 10 for LUTs with 0, 1, 2, 3 or 4 active inputs. The required data for each fault injection consists on the 16 bits vector LUT information, modified in order to inject the fault whenever the activation conditions are present.

In order to inject a LSA-*v* fault (*v* = 0 or *v* =1) at the output of a LUT, all 16 bit positions of the LUT reconfiguration vector must be set to the logic value *v*. The reconfiguration vector that corresponds to the LUT input *a* stuck at value *v* is obtained by copying the values y_{vbcd} to $y_{\neg vbcd}$ for each *bcd* combination. For instance, the vector for the fault input *a* LSA-0 is obtained by copying y_{0000} to y_{1000}, y_{0001} to y_{1001}, y_{0110} to y_{1110}, …, y_{0111} to y_{1111}.

After knowing the 16 bit programming vectors for each fault, fault collapsing can be performed efficiently by grouping the faults with identical programming vectors.

2.3 – Combination Stuck-At (CSA) Fault Model

The main purpose a fault simulation tool is to evaluate test quality, through the fault coverage metrics. Hence, even for non-FPGA implementations, a fundamental issue is the correspondence between LUT faults and the fault universe that we consider relevant and we want to guarantee detection (or robustness) in the final design implementation.

The detection of the LSA faults at every LUT terminal does not guarantee that every realistic fault on the logic structure that may implement the LUT function is detected. To

illustrate this, consider the LUT functionality *y=a.b+c.d*. Table 1 shows all the vectors required for the injection of all the LSA faults of this LUT (from column a0=b0 to column y0). Notice that two fault pairs were collapsed using the methodology presented in the previous section *a* LSA0 and *b* LSA0; *c* LSA0 and *d* LSA0. The shaded positions of the table indicate LUT input combinations (LUT test vectors) that activate the corresponding fault. One possible test sequence that activates all LSA faults in this LUT is the one with vectors number 7, 10, 12 and 15 (see the 1st column of Table 1).

One possible logic structure that implements the functionality *y=a.b+c.d* is presented in Fig. 1. Consider a possible bridging fault between nodes *e* and *f*. The columns *e* and *f* of Table 1 show the fault free values of these signals. The values forced by two transistors in a standard nand gate are marked bold. The last column of Table 1 shows the values for *y* assuming that, when in the conflict, two transistors in parallel force the dominant logic value. Other conflicts without a pull-up network of two transistors are marked as unknown.

This bridging fault (e,f) is not modeled by any of the LUT LSA faults. The LUT programming vector for the fault *e* LSA0 is presented in the last column of Table 1. As it can be seen, vectors 4 or 13 are the only ones that guarantee the detection of this fault and they are not included in the test sequence identified for detection of all LSA faults in the LUT terminals.

Table 1: Fault injection vectors for LSA faults at the LUT terminals and one internal node LSA0. LUT implements y=a.b+c.d.

Vector #	a	b	c	d	y=ab+cd	a0=b0	c0=d0	a1	b1	c1	d1	y1	y0	e	f	y=bri(e,f)
1	0	0	0	0	0	0	0	0	0	0	0	1	0	**1**	**1**	0
2	0	0	0	1	0	0	0	0	0	1	0	1	0	**1**	1	0
3	0	0	1	0	0	0	0	0	0	0	1	1	0	**1**	1	0
4	0	0	1	1	1	1	0	1	1	1	1	1	0	**1**	0	0
5	0	1	0	0	0	0	0	1	0	0	0	1	0	1	**1**	0
6	0	1	0	1	0	0	0	1	0	1	0	1	0	1	1	0
7	0	1	1	0	0	0	0	1	0	0	1	1	0	1	1	0
8	0	1	1	1	1	1	0	1	1	1	1	1	0	1	0	x
9	1	0	0	0	0	0	0	0	1	0	0	1	0	1	**1**	0
10	1	0	0	1	0	0	0	0	1	1	0	1	0	1	1	0
11	1	0	1	0	0	0	0	0	1	0	1	1	0	1	1	0
12	1	0	1	1	1	1	0	1	1	1	1	1	0	1	0	x
13	1	1	0	0	1	0	1	1	1	1	1	1	0	0	**1**	0
14	1	1	0	1	1	0	1	1	1	1	1	1	0	0	1	x
15	1	1	1	0	1	0	1	1	1	1	1	1	0	0	1	x
16	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	1

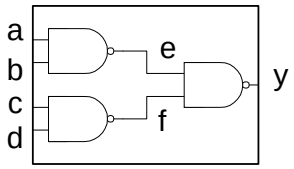

Fig. 1: Possible structural implementation of y=a.b+c.d

In order to achieve a LUTs coverage metric that better correlates with realistic fault coverage a new fault model is proposed: **C**ombination **S**tuck **A**t (CSA). The number of faults, for this model, is identical to the possible combinations of LUT active inputs. Each fault is injected by inverting only the LUT bit position that corresponds to one input *combination*. The total coverage of CSA faults corresponds to the exhaustive LUT functionality test. Thus, CSA is a *functionality* driven fault model instead of a *line* driven fault model. Note that CSA test requires half the vectors needed for testing each LUT position with both logic values (exhaustive LUT testing).

Table 2 compares the number of CSA faults with the number of LSA faults for different LUT types. The column “LSA colap.” presents an approximate number of

collapsed LSA faults based on the experiments that are reported in section 4. This Table shows that, when the CSA fault model is used, it leads to an increase in the fault list size, specially for LUTs with 4 active inputs. As these are the most used LUTs, this implies an increase of near 90%. This increase reflects linearly in the fault simulation time, since in HFS faults are injected consecutively. Nevertheless, since partial reconfiguration based HFS is very fast, this increase in the fault list size does not make it less attractive, as it will be demonstrated in section 4.

Table 2: Number of LSA faults and CSA faults.

LUT type	LSA faults	LSA colap.	CSA faults
LUT0	2	1	1
LUT1	4	2	2
LUT2	6	4	4
LUT3	8	5	8
LUT4	10	8	16

3 – Hardware Fault Simulation (HFS) Tool - f²s

A new tool for **F**PGA based **f**ault **s**imulation (f²s) was developed. f²s takes one unique input which is the binary file with the complete FPGA configuration. From this file, f²s identifies the target device for which the mapping was performed. This identification is mandatory in order to know the number of rows and columns for LUT bit positioning. LUT extraction is carried out as described in section 2.1 and the fault list is collapsed as described in section 2.2. After fault collapsing, f²s creates a report with all the LUTs in use in the FPGA, the original fault list, the collapsed fault list and the estimated reconfiguration times for different hardware interfaces. Fig. 2 shows this report for the s5378 ISCAS'89 benchmark circuit, used in section 4 as a test vehicle. Afterwards, partial reconfiguration bit files are generated for each fault in a sequence where each bit file reprograms only the minimum number of frames (smallest amount of data that can be read or written with a single command) required to un-inject the previous fault and inject the following one. Both single and multiple fault injection are supported. Debugging and validation of the tool was carried out using a JTAG 200 kb parallel III port and the experimental values obtained for the programming times matched the estimated values. Since the current version of the tool does not support any type of vector formatting or application, it is more suitable for BIST quality evaluation purposes.

```
BIT_FILE_NAME:  = s5378.ncd                                   BIT_TIME_STRING:  = 18:01:59
BIT_PART_STRING:  = v2000ebg560                               BIT_IMAGE:  size 1269956
BIT_DATE_STRING:  = 2003/01/16
FAULTS STACK AT ZERO/ONE at LUT´s INPUTS / OUTPUTS / SELECT RAM BITS
LUT -- LOCATION , TYPE , CONTENTS AND FAULT TYPE TO INJECT ( equivalent faults , between – and separated by / )
row=49      col=19    slice=1    lut=G     type=2    vector=0 0 0 0 0 0 1 1 0 0 0 0 0 0 1 1
faults= -- O_S_1/I1_S_0/I2_S_0 – O_S_0 – I2_S_1 – I1_S_1
row=61      col=32    slice=0    lut=G     type=2    vector=1 1 0 0 1 1 0 0 1 1 1 1 1 1 1 1
faults= -- O_S_1 – O_S_0/I0_S_1/I2_S_0 – I2_S_1 – I0_S_0
…
row=55      col=73    slice=0    lut=G     type=4    vector=0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
faults= -- O_S_1/I0_S_0/I1_S_0/I2_S_0/I3_S_1 – O_S_0 – I1_S_1 – I2_S_1 – I0_S_1 – I3_S_0
lut capacity   38400                                          lut type 1    0     0
lut not used   37996    98.9479166666667                       lut type 2    66    0.171875
fauls to detect 3610                                          lut type 3    83    0.216145833333333
fauls to inject 2707   0.749861495844875                      lut type 4    255   0.6640625
TIME (hour:min:sec) to program and inject     0:6:51    (based on 200 Kb parallel III port speed – jtag)
TIME (hour:min:sec) to program and inject     0:0:16    (based on 5 Mb parallel IV port speed – jtag)
TIME (hour:min:sec) to program and inject     0:0:1     (based on 66 Mb usb port speed – serial)
```

Fig.2: Typical fault list report of f²s.

4 – Results

Software and Hardware FS can be compared in many different ways. Here, the focus is to spot the trade-off associated with the fundamentals of the two approaches: SFS costs increase steadily with circuit complexity, test length and fault list size, especially for sequential circuits. HFS has a significant initial cost, associated with FPGA synthesis, and configuration, and low cost in the hardware simulation process itself[1]. In order to identify the conditions for which HFS is advantageous as compared to SFS, two ISCAS'89 circuits have been used, with different test lengths, as test vehicles. The complexity of these circuits, as well as the fault lists sizes, are shown in Table 3. From Table 3 it is clear that LSA fault model on LUT terminals is collapsed to a very reduced fault list when compared to the collapsed fault list used in SFS. This reinforces the idea that, for HFS in FPGA, the LSA fault model at LUT terminals leads to incomplete coverage analysis with limited value for a generic LUT logic implementation. The proposed CSA fault model leads to fault list sizes similar to the ones used in SFS. Table 4 shows the LSA fault collapsing results for the s5378 circuit obtained with the new `f²s` tool. A set of random test vectors has been used for the SFS and HFS experiments.

Table 3: ISCAS'89 characteristics mapped in a Virtex FPGA and corresponding fault list sizes.

Circuit	# LSA faults (after coll.)	# LUTs in use	# of free LUTs 2000E	# LSA faults HFS	# LSA faults HFS (after coll.)	# CSA faults
s5378	4441	404	37996	3610	2707	5008
s13207	9763	795	37605	7126	5434	9862

Table 4: Number of faults (after collapsing) for the LUTs of s5378 benchmark circuit .

LUT type	# faults						
	4	5	6	7	8	9	10
LUT2	63		3				
LUT3		40	24	7	14		
LUT4			54	47	118	11	24

Software fault simulation was carried out using a widely used, commercial fault simulation tool on a Sun ultra10/440 workstation with 1 GB RAM. The metrics for comparing HFS and SFS is the *simulation time*, namely the HFS time, t_{HFS}, and the SFS time, t_{SFS}, The SFS time, t_{SFS}, is computed by the commercial EDA tool. The HFS time, t_{HFS}, is estimated as

$$t_{HFS} = t_{conf} + \#\,faults \times \left(t_{reconf} + \frac{\#vectors}{f_{HFS}} \right) \qquad (1)$$

where t_{conf} is the sum of the FPGA synthesis and configuration times, t_{reconf} is the time required for reconfiguration (which is zero in the case of a unique configuration) and f_{HFS} is the frequency of vector application in the hardware. t_{comp} is obtained after synthesis and mapping of the circuit using the Xilinx™ `xst` synthesis tool. The `f²s` tool analyses the bit stream and, taking into account the exact number of frames that are required to reprogram during the fault simulation process, reports the t_{reconf} for different hardware interfaces. Figs. 3 and 4 show fault simulation times for the two ISCAS'89 benchmark circuits, using f_{HFS} = 50MHz on a Xilinx™ Virtex2000e FPGA embedded on a Celoxica RC1000-PP board, without fault dropping in HFS. The curves marked with "LSA 200kb" and "CSA 200kb" refer, respectively, to LSA and CSA fault injection using a JTAG 200 kb parallel III port.

[1] We prefer to use the term *emulation* for the hardware simulation of a fault-free device.

The curve marked with “LSA 5Mb” refer to LSA fault injection using a JTAG 5 Mb parallel IV port.

From these figures it is clear that HFS without partial reconfiguration (“unique configuration”, in Figs. 3 and 4 [9]) has advantage over SFS if the number of test vectors exceeds a threshold that mainly depends on t_{comp}. Of course, the slope of t_{SFS} depends on the computer resources of the machine running the software fault simulation tool. HFS with fault injection by partial reconfiguration is faster than SFS even for a small number of test vectors. This advantage of HFS with partial FPGA reconfiguration is evident even if the proposed CSA fault model is used.

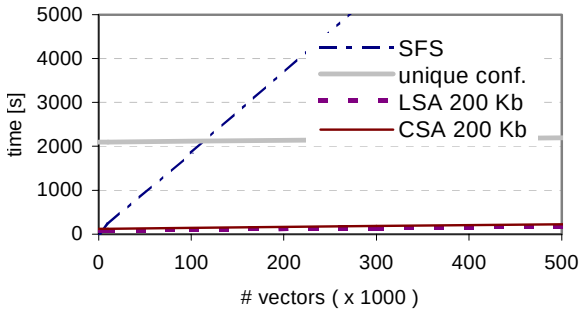

Fig. 3: HW and SW fault simulation times for the s5378 benchmark circuit.

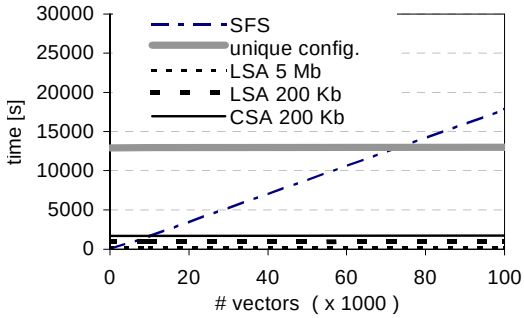

Fig. 4: HW and SW fault simulation times for the s13207 benchmark circuit.

5 – Conclusions

It has been shown in this work that FPGA-based HFS using partial reconfiguration is rewarding, as an alternative to SFS, constraining fault simulation costs (and thus time to market), even for a limited number of test vectors. Usual clock frequencies make HFS time two orders of magnitude less dependent on the number of vectors than SFS, for the examples studied. The advantages of partial reconfiguration-based HFS has been validated through the software and hardware fault simulation of ISCAS’89 benchmark circuits.

Fault simulation using only the LSA fault model at LUT terminals, collapsible to a very reduced fault list, does not provide a complete coverage measure of LUT exercise. Hence, a new fault model, CSA, driven towards the LUT functionality, has been proposed. The CSA fault model has been shown to increase the fault list size to the order of magnitude of the SFS fault list, while not making HFS less attractive. It is expected that CSA fault coverage on FPGA implementations will exhibit useful correlation with realistic fault coverage on synthesized logic structures, to be mapped in IP cores. This has not been extensively demonstrated in the present paper, as the goal was to introduce the novel HFS methodology and tool (for LUT-based FPGAs); however, this will be reported elsewhere.

The work has been carried out using a developed software tool, f²s, which takes as unique argument the binary file for FPGA configuration, identifies the target FPGA, extracts the LUT usage and the faults and generates minimal reconfiguration bit files suitable for fault injection. Future work deals with the automation of vector formatting and HFS control for particular FPGA interfaces (Celoxica RC1000-PP board) in order to make f²S usable as a standard HFS tool. Work is in progress, and will be reported in the future.

Acknowledgement

This work has been partially funded by FCT (Portugal), POCTI/ESE41788/2001.

References

[1] M.L. Bushnel, V.D. Agrawal, “Essentials of Electronic Testing for Digital Memory and Mixed-Signal VLSI Circuits”, Kluwer Academic Pubs., 2000.

[2] T. M. Niermann, W. T. Cheng, J. H. Patel, “PROFS: A fast, memory-efficient sequential circuit fault simulator”, IEEE Trans. Computer-Aided Design, pp.198-207, 1992.

[3] E. M. Rudnick, J. H. Patel, “Overcoming the Serial Logic Simulation Bottleneck in Parallel Fault Simulation”, Proc. of the IEEE International Test Conference (ITC), pp. 495-501, 1997.

[4] F.M. Gonçalves, M.B. Santos, I.C. Teixeira and J.P. Teixeira, "Design and Test of Certifiable ASICs for Safety-critical Gas Burners Control", Proc. of the 7th. IEEE Int. On-Line Testing Workshop (IOLTW), pp. 197-201, July, 2001.

[5] R.W.Wieler, Z. Zhang, R. D. McLeod, “Simulating static and dynamic faults in BIST structures with a FPGA based emulator”, Proc. of IEEE Int. Workshop of Field-Programmable Logic and Application, pp. 240-250, 1994.

[6] K. Cheng, S. Huang, W. Dai, “Fault Emulation: A New Methodology for Fault Grading”, IEEE Trans. On Computer-Aided Design of Integrated Circuits and Systems, vol. 18, no. 10, pp1487-1495, October 1999.

[7] Shih-Arn Hwang, Jin-Hua Hong and Cheng-Wen Wu, “Sequential Circuit Fault Simulation Using Logic Emulation”, IEEE Transations on Computer-Aided Design of Integrated Circuits and Systems, vol. 17, no. 8, pp. 724-736, August 1998.

[8] P.Civera, L.Macchiarulo, M.Rebaudengo, M.Reorda, M.Violante, “An FPGA-based approach for speeding-up Fault Injection campains on safety-critical circuits”, IEEE Journal of Electronic Testing Theory and Applications, vol. 18, no.3, pp. 261-271, June 2002.

[9] M. B. Santos, J. Braga, I. M. Teixeira, J. P. Teixeira, "Dynamic Fault Injection Optimization for FPGA-Based Harware Fault Simulation", Proc. of the Design and Diagnostics of Electronic Circuits and Systems Workshop (DDECS), pp. 370-373, April, 2002.

[10] Miron Abramovici, Prem Menon, “Fault Simulation on Reconfigurable Hardware” , *IEEE Symposium on FPGAs for Custom Computing Machines*, pp. 182-190, 1997.

[11] M. Abramovici, P. R. Menon, D. T. Miller, “Critical Path Tracing: An Alternative to Fault Simulation”, IEEE Design Automation Conference, pp. 468 - 474 , 1984.

[12] L. Burgun, F. Reblewski, G. Fenelon, J. Barbier, O. Lepape, “Serial fault simulation”, Proc. Design Auomation Conference, pp. 801-806, 1996.

[13] L.Antoni, R. Leveugle, B. Fehér, “Using Run-Time Reconfiguration for Fault Injection in Hardware Prototypes”, IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, pp.405-413, October 2000.

[14] L.Antoni, R. Leveugle, B. Fehér, “Using Run-Time Reconfiguration for Fault Injection Applications”, IEEE Instrumentation and Measurement Technology Conference, vol. 3, pp.1773-1777, May 2001.

[15] L.Antoni, R. Leveugle, B. Fehér, “Using Run-Time Reconfiguration for Fault Injection in Hardware Prototypes”, IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, pp. 245-253, 2002.

[16] S. Guccione, D. Levi, P.Sundararajan, “Jbits: A Java-based Interface for Reconfigurable Computing”, Proc. of the 2nd Military and Aerospace Applications of Programmable Devices and Technologies Conference (MAPLD), pp. 27, 1999.

[17] E. Lechner, S. Guccione, “The Java Environment for Reconfigurable Computing”, Proc. of the 7th International Workshop on Field-Programmable Logic and Applications (FPL), Lecture Notes in Computer Science 1304, pp.284-293, September 1997.

[18] P.Sundararajan, S.Guccione, D.Levi, “XHWIF: A portable hardware interface for reconfigurable computing”, Proc. of Reconfigurable Technology: FPGAs and Reconfigurable Processors for Computing and Communications, SPIE 4525, pp.97-102, August 2001.

[19] F. Brglez, D. Bryan, K. Kominski, "Combinational Profiles of Sequential Benchmark Circuits", Proc. Int. Symp. on Circuits and Systems (ISCAS), pp. 1229-34, 1989.

[20] Xilinx Inc., “Virtex-E 1.8V Field Programmable Gate Arrays”, Xilinx DS022, 2001.

[21] Xilinx Inc., “Virtex Series Configuration Architecture User Guide”, Application Note: Virtex Series, XAPP151 (v1.5), September 27, 2000.

[22] H. Cha, E. Rudnick, J.Patel, R. Iyer, G.Shoi, “A Gate-Level Simulation Environment for Alpha-Particle-Induced Transient Faults”, IEEE Transactions on Computers, vol. 45, no.11, pp1248-1256, November 1996.