BUILT-IN SELF-TEST PREPARATION IN FPGAS

Abílio Parreira
aemp@netvisao.pt

J. P. Teixeira
jct@inesc-id.pt

Marcelino B. Santos
marcelino.santos@inesc-id.pt

IST / INESC-ID, R. Alves Redol, 9, 1000-029 Lisboa, Portugal

Abstract. *This paper addresses the problem of Test Effectiveness (TE) evaluation of a FPGA BIST solution, through Hardware Fault Simulation (HFS). A novel HFS technique is proposed, that efficiently, using partial reconfiguration, ascertain (or not) the BIST. The proposed methodology can be particularly useful for signature fault dictionary building and for applications in which multiple fault injection is mandatory. HFS is performed using local partial reconfiguration with small binary files. The limitation of injecting faults in Look-Up Tables (LUTs) is evaluated as a sampling technique. The methodology is fully automated and our results show that it can be orders of magnitude faster than software or fully reconfigurable hardware fault simulation.*

1 Introduction

Designs for Testability (DfT) techniques are mandatory for cost-effective product development [1]. The market of reconfigurable systems is becoming a significant share. Hence, many new products use reconfigurable devices, namely Field Programmable Gate Arrays (FPGAs). Stand-alone or embedded FPGA cores in Systems on a Chip (SoC) need to be tested. Dependable systems require that, during product lifetime, FPGA components may be tested, in particular for the target programmed functionality. Such system specification makes the use of Built-In Self-Test (BIST) technology very attractive. In fact, lifetime testing using BIST may allow a low cost solution, and by using at-speed testing, it enables the detection of *dynamic* faults, which are relevant in DSM (Deep Sub-Micron) semiconductor technologies [1].

BIST technology routinely uses PR (Pseudo-Random) TPG (Test Pattern Generators), like LFSRs, and signature analyzers, like MISRs. Random pattern resistant faults may require refined approaches, e.g., as weighted PR, or re-seeding techniques [2]. The modules under self-test may be combinational, sequential or reconfigured in test mode. Nevertheless, *test quality* needs to be ascertained, during the design phase. Four quality metrics can be defined: TE (Test Effectiveness), TO (Test Overhead), TL (Test Length, the number of test vectors in the test session) and TP (Test Power). The necessary condition is TE, i.e., the ability of a given test pattern to uncover likely defects, induced during IC manufacturing or during lifetime operation and aging. Defects coverage is usually evaluated through the FC (Fault Coverage) metrics, using a fault model, such as the single Line Stuck-At (LSA) fault model. Typically, BIST solutions lead to low TO (additional hardware resources / speed degradation), large TL (compensated by at-speed vector application) and moderately high TP (Power consumption required for the BIST session).

The major issue addressed in this paper is: *how can we, in a cost-effective way, evaluate the TE of a given BIST solution?*

During the design phase, test quality may be assessed by Fault Simulation (FS). For complex devices, FS may be a very costly process, especially for sequential circuits. In fact, circuit complexity, TL and fault list size may lead to a large computational effort. Although many efficient algorithms have been proposed for SFS (Software Fault Simulation) (see, e.g., [3] and [4]), for complex circuits it is still a very time-consuming task and can significantly lengthen the time-to-market. Moreover, observability in embedded BIST is not for each vector, but only for each signature captured in the MISR, after all test vectors are applied. This fact may compromise fault dropping, routinely used to severely restrict SFS costs.

FS can be implemented in software or hardware [1]. The ease of developing software tools for FS (taking advantage of the flexibility of software programming) made SFS widely used. However, the recent advent of very complex FPGAs components created an opportunity for HFS (Hardware Fault Simulation), which may be an attractive solution for at least a subset of practical situations. In fact, BIST TE evaluation may require a heavy computational effort in fault simulation. Long test sessions (large TL) are needed to evaluate systems composed of several modules that have to be tested simultaneously in order to evaluate aliasing. One example is functional BIST in a SoC with data compression cores being used as part of the signature compression logic. Another FS task not efficiently performed by software tools is *multiple fault simulation*, mainly because of the enormous number of possible fault combinations. However, multiple fault simulation may become mandatory, namely in the certification process of safety-critical applications [5]. HFS may cope with these issues. In particular, *if the FPGA design with BIST, under evaluation, is programmed in the target FPGA component, could we use it for fault injection, and FC computation?*

The purpose of this paper is to present a highly efficient HFS methodology and tool for BIST preparation on a target FPGA. The methodology uses an efficient partial reconfiguration technique for fault injection, by deriving very small bit files for fault injection. These bit files are obtained from the full configuration bit file, by means of its direct manipulation, without requiring any additional software tools. A flexible fault model is introduced: the HFS reads the LUT faulty behavior from a user-defined file. Moreover, LUT fault injection effectiveness is evaluated as a fault sampling technique.

This paper is organized as follows: in section 2, previous work is reported;. In section 3, the LUT extraction from the bit file and partial reconfiguration is explained. Section 4 presents the fault model. Section 5 describes experimental results that evaluate the fault sampling and compare SFS and HFS. Finally, section 6 presents the conclusions of this work.

2 Previous work

Different HFS approaches using FPGAs have been proposed in the literature to overcome the difficulties with SFS for sequential circuits.

Dynamic fault injection, using dedicated extra hardware, and allowing the injection of different faults without reconfiguring the FPGA, was proposed in [6-9]. The additional hardware proposed for implementing dynamic fault injection uses a Shift Register (SR) whose length corresponds to the size of the fault list. Each fault is injected when the corresponding position in the SR has logic “1”, while all other positions have logic “0”. Upon initialization, only the first position of the SR is set to “1”. Then, the “1” is shifted along the SR, activating one fault at a time. This technique was further optimized in [10]. However, a major limitation of this technique is the fact that the added hardware increases with the number of faults to inject, which limits the size of the circuits that can be simulated.

In [7], it is shown that parallelism is possible by injecting independent faults at the same time. This parallelism is limited to faults in different propagation cones; however, the reported speedup is only 1.36 times the pure serial FS.

In [11], a new approach that included a backward propagation network to allow critical path tracing [12] is proposed. This information allows multiple faults to be simulated for each test

vector; nevertheless, it also requires heavy extra hardware. Only combinational circuits have been analyzed.
A serial FS technique that requires only partial reconfiguration during logic emulation was proposed in [13], showing that no extra logic need be added for fault injection purposes. The authors show that HFS can be two orders of magnitude faster than SFS, for designs over 100,000 gates.

More recently, other hardware fault injection approaches were proposed in [14-16] using the JBITS [17] interface for partial FPGA reconfiguration. The JBITS software can achieve effective injection of faults on Look-Up-Tables (LUTs) [14,15] or erroneous register values [16]. These papers do not report any actual results on partial FPGA reconfiguration times for fault injection - only predictions are given. Moreover, the approach that uses JBITS requires the Java SDK 1.2.2 [18] platform and the XHWIF [19] hardware interface.

3. LUTs Extraction and Fault Injection

3.1 LUT Extraction

Xilinx Virtex [20] and Spartan FPGA components were used in this work, due to the fact that partial reconfiguration of these components is possible and documented [21]. The proposed methodology is an extension of the work reported in [22]. As shown in [21], the binary file for configuration of these FPGAs can be divided in 6 major parts:

1. Header with project and target FPGA id.
2. Enable program mode.
3. CLB configuration.
4. BRAM configuration.
5. Error detection code (crc).
6. Enable normal operation mode.

Each configuration file (total or partial) consists on sequences of commands and frames. Each command contains two 32 bit word: the first word is an operation code. The second word is an operand. Frames are the smallest amount of data that can be read or written with a single command; frame size depends on the specific FPGA number of rows: 18 times the number of rows, rounded up to a multiple of 32. CLBs (Configurable Logic Blocks) are the building blocks for implementing custom logic. Each CLB contains two slices. Each slice contains two 4 input LUTs, 2 flip-flops and some carry logic. In order to extract the LUT's configuration, the software tool developed analyses the part of the frames that describes the CLBs configuration, as depicted in Figure 1.

Each LUT can be used to implement a function of 0, 1, 2, 3 or 4 inputs. The number of inputs relevant for each used LUT must be identified, for the target functionality, in order to include the corresponding faults in the fault list.

Since each LUT has 4 inputs, the LUT contents consist of a 16-bit vector, one vector for each combination of the inputs. Let y_{abcd} be the output value for the combination of input values *abcd*. The LUT configuration 16-bit vector can be denoted y_{0000}, y_{0001}, y_{0010}, …, y_{1111}, where each bit position corresponds to the LUT output value for the respective combination of the inputs i_3, i_2, i_1 and i_0. If one input has a fixed value, then an 8 bit vector is obtained. For instance, if we have always i_3=0, then the relevant 8 bit vector is y_{0000}, y_{0001}, y_{0010}, y_{0011}, y_{0100}, y_{0101}, y_{0110}, y_{0111}.

After retrieving the information of each LUT from the bit file, the relevance of each input i_x (x=0,1, 2 and 3) is evaluated, by comparing the 8-bit vectors corresponding to i_x=0 and i_x=1. If these two vectors are identical, then the input i_x is not active. This is how we extract the LUT types, e.g., LUT2, LUT3 or LUT4, according to their number of relevant inputs. In this LUT extraction process, the nets associated with the LUT inputs and output are also identified, in order to establish a link between the faults to inject and the design.

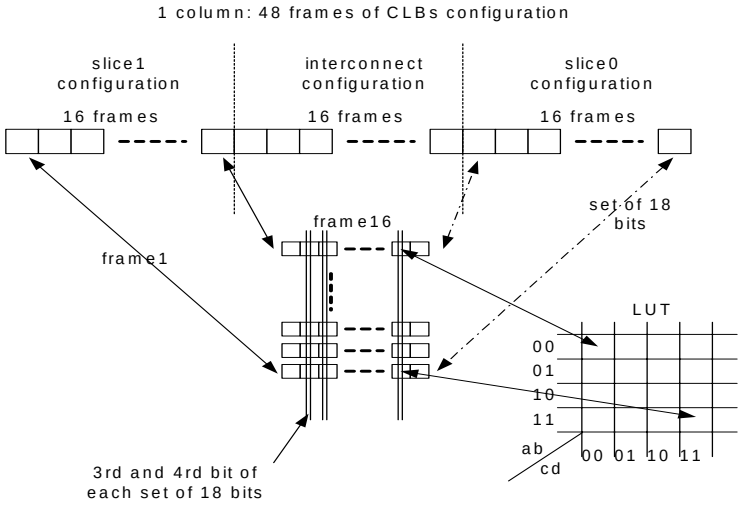

Figure 1 – Positions in the frames of the LUT configuration bits.

3.2 Fault Injection and Simulation

After LUT extraction, fault injection is carried out by partial reconfiguring the FPGA. The assumed fault model is a user's defined set of LUT faults, as we realize that a very efficient partial reconfiguration technique could be developed. Fault injection triggers a modification of individual LUT contents, modifying the LUT contents (Karnaugh map) according to the assumed fault model. The new approach supports a generic fault model for LUTs, and is presented in section 4.1. Partial reconfiguration bit files, targeting each fault, are generated in sequence. In this way, each bit file reprograms only the minimum number of frames required to un-inject the previous fault and inject the next one. The binary file for partial reconfiguration requires the faulty frames and a more restricted number of commands, that is not clear in [21], but we have identified as:

1. · Header - 32 bits "1".
2. · Synchronization command.
3. · CRC reset command.
4. · FLR – specification of the frame size.
5. · COR – start-up enable command.
6. · MASK – enable writing the CTL register.
7. · ASSERT HIGH
8. · FAR - address of the initial CLB frame.
9. · write – enable configuration writing.
10. · FDRI – number of configuration words to send.
11. · CLBs configuration words
12. · CRC
13. · LFRM – type and address of the last frame
14. · FDRI – number of configuration words to send for the last frame.
15. · Configuration words for the last frame.
16. · START-UP – start-up enable.
17. · CTL – Memory read enable and configuration pins use specification.
18. · CRC
19. · Termination bit

This partial reconfiguration is repeated during the FS process. At present, this process is automated only for BIST, according to the target application described in this work. However, other HFS strategies may be considered, namely emulation of the FS process of a target device. In order to perform FS, the developed tool sends a start BIST commands and waits for BIST end to read the signature in the multiple input shift register (MISR). At present, this interaction between the software that controls the fault simulation process and the FPGA is accomplished using the JTAG 200 Kb parallel III port. Xilinx USER1 instruction is used as the start BIST command. USER2 instruction is the read signature command. The validation of the tool was carried out in a *DIGILAB 200 E* board with a Xilinx Spartan 200 E FPGA. However, the developed software tool supports every board with JTAG interface with Virtex or Spartan FPGAs, including the E type.

The developed tool delivers a fault simulation report. This report includes, for each fault, the target LUT type and location, equivalent faults, detection result and MISR signature. The LUT inputs and output are also identified, using the information from the LUT extraction.

4. Fault Models

4.1 LUT Fault Models

In order to inject an active fault, the LUT content must be changed. This change depends on the fault to inject and on the LUT fault free information. LUTs with identical fault free configurations require fault injections with the same faulty LUT contend. In order to increase efficiency, the faulty LUT contend is pre-computed for each possible fault in the fault-free LUT configuration. The LUT is viewed as a 16 bits vector. The definition of fault model is made in a file that associates one or several LUTs faulty contents to each possible combination of the 16 bits that correspond to the fault free LUT configuration. The simulation tool, after loading this file, can easily inject all the faults associated to each LUT configuration, previously obtained as described in section 3.1.

An auxiliary tool was developed to generate the faulty LUT configurations for different fault models: single and multiple LSA. As an example, 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-1 is obtained, as illustrated in Figure 2, by copying y_{1000} to y_{0000}, y_{1001} to y_{0001}, y_{1110} to y_{0110}, …, y_{1111} to y_{0111}. It is a major advantage to perform this calculation only once and prior to fault simulation not only because it is more efficient but also because it adds flexibility to the fault model.

Following the procedure of obtaining these faulty configurations, they are compared. Fault collapsing is performed when different faults result in identical faulty configurations.

In Table 1, the original (and the collapsed) number of faults is given, for the single and multiple LSA fault model, for ISCAS'85 and 89 benchmark circuits [23] c7552, s5378, s13207 and for a clone from ARM7 [22]. Multiple LSA faults includes all combinations of 2, 3 and 4 faults in the LUT inputs. Fault collapsing is more relevant, as it can be seen, for multiple faults model.

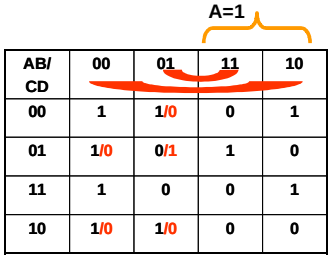

Figure 2: Computing the 16 bits for LUT injection of fault "input *a* LSA-1".

	Single LSA		Multiple LSA	
	total	collapsed	total	collapsed
c7552	9374	7422	53566	17331
s5378	6168	4479	35100	10065
s13207	14232	10269	80180	23020
ARM7	93710	70576	624676	185473

Table 1: Original and collapsed number of faults

4.2 LUT Faults vs. CLB Faults

Fault injection is carried out in LUTs only. What confidence can we have in a test that ensures a high coverage of LUTs faults will also cover the remaining faults? In order to answer to this question, one can view LUT faults as a *sampling technique*, as compared with the complete fault set. Fault sampling techniques have been widely used and documented (e.g., [1,25]), showing that the designer does not need to simulate the whole fault set to obtain a FC figure, close to the global FC figure. Therefore, in this particular application and for the target functionality, we need to evaluate how representative is the fault sampling that consists of taking only the LUTs faults in a CLB.

For this analysis, the fault list considered for the CLB consists on the LSA faults on each CLB logic element output and inputs. Faults in the selection lines of multiplexers with constant values

are not considered, as they are considered as routing faults. LSA faults in routing elements are assumed to be collapsed to faults at lines of logic elements.

For the considered FPGA components, each CLB has LUTs and 11 additional logic elements. Table 2 shows the number and type of logic elements, the corresponding number of I/Os, the original number of faults and the collapsed number of LSA faults. Since the number of collapsed faults in a LUT with 4 inputs is 8, LUT i/o LSA faults represent only 16/(16+78)=17% of the CLB LSA faults.

However, BIST TE must evaluate only the fault coverage in the circuitry in used by the specific configuration, i.e., on the target functionality. As shown in the next section, the use of LUT is dominant and typically fault samplings are over 70%. Such significant fault sample size allow us to conclude that the sample coverage (using only LUT faults) will lead to a good estimate of the CLB fault coverage, within a narrow confidence interval.

5. Results

The FPGA implementation of the ARM7 [22] core results, in terms of the occupation of the resources in a Spartan 600E Xilinx FPGA are listed in Table 3.

# cells / CLB	Cell name	# inputs	# outputs	# faults	# collap. faults	# CLB col. faults
2	and	2	1	6	4	8
1	MUXCY	6	2	16	12	12
1	MUXF5	3	1	8	6	6
1	MUXF6	3	1	8	6	6
2	FF	5	1	12	10	20
2	XORCY	2	1	6	4	8
1	DGEN	3	1	8	8	8
1	WSGEN	4	1	10	8	10
Total number of faults					58	78

Table 2: Collapsed faults in CLB logic elements, excluding LUTs in Spartan and Virtex Xilinx FPGAs.

		Original		Collapsed	
Cell name	# cells	# faults / cell	# faults	# faults / cell	# faults
LUT1	84	4	336	2	168
LUT1_L	15	4	60	2	30
LUT2	527	6	3162	4	2108
LUT2_D	8	6	48	4	32
LUT2_L	85	6	510	4	340
LUT3	2619	8	20952	5	13095
LUT3_D	62	8	496	5	310
LUT3_L	17	8	136	5	85
LUT4	6366	10	63660	8	50928
LUT4_D	110	10	1100	8	880
LUT4_L	325	10	3250	8	2600
MUXCY	474	16	7584	12	5688
MUXF5	824	8	6592	6	4944
MUXF6	28	8	224	6	168
XORCY	403	6	2418	4	1612
FDC	1349	8	10792	8	10792
FDCE	525	10	5250	10	5250
FDCP	30	10	300	10	300
FDP	16	8	128	8	128
FDPE	24	10	240	10	240
Total # faults		127238		99698	
# LUTs faults		93710		70576	
Fault sampling %		73.6		70.8	

Table 3: Logic elements allocated for the ARM7 implementation.

Table 3 presents the original and the collapsed fault list size for each logic element and the global fault list size for the circuit. For this example, using only LUT faults is equivalent to sample 70.8% of the collapsed fault list. The huge difference from the worst-case value evaluated in the previous section is due to the majority of LUTs as processing logic element. This is also the case for all the circuits that we have studied. Table 4 shows the fault sampling values for three additional benchmark circuits [23].

The effectiveness of the proposed HFS technique critically depends on the simulation times. Figures 3 and 4 show the fault simulation times using JTAG 200 Kb parallel III port. From these

figures it is clear that HFS is advantageous over SFS after a reduced number of test vectors. After 100000 vectors, HFS is orders of magnitude faster than SFS. Moreover, from Figure 4, it can be seen that the new HFS tool simulates a clone from the ARM processor with 70576 faults and one million test vectors in nearly two hours. The flat initial part of all curves in Figure 4 is due to the fact that there is a minimum amount of time required to process each fault information. If the BIST session applies a small number of vectors and runs fast, the fault simulation time is not limited by the BIST time.

Circuit	c7552	s5378	s13207	ARM7
Total	13836	9592	23294	127238
# LUT faults	9374	6168	14232	93710
Total collapsed	10029	6308	14882	99698
# LUT faults collapsed	7422	4479	10269	70576
Fault sampling [%]	74,0	71,0	69,0	70.8

Table 4: Fault sampling resulting from injecting faults in LUTs.

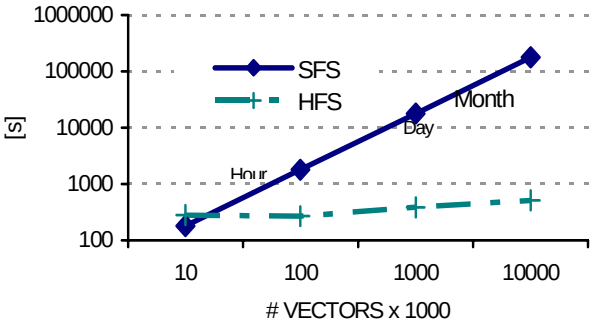

Figure 3: Hardware vs. software fault simulation time for the s13207 benchmark.

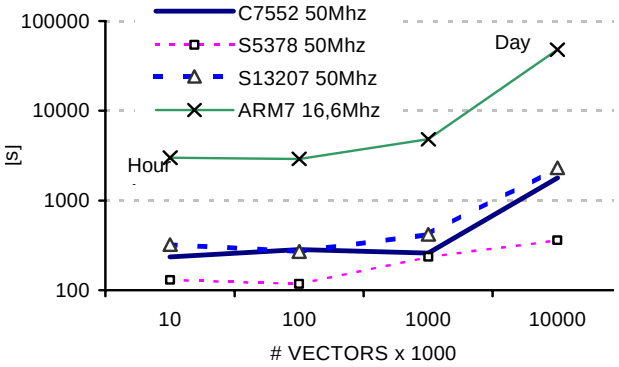

Figure 4: Hardware fault simulation time for different benchmark circuits.

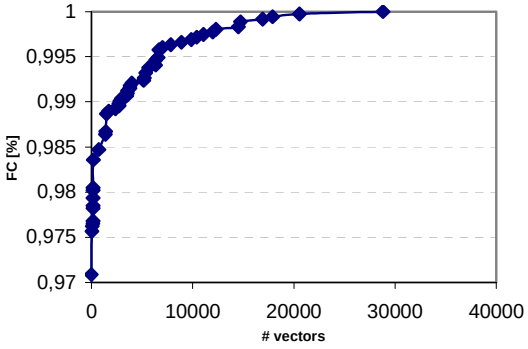

Figure 5: Fault coverage evolution for the c7552.

Fault coverage results can also add in deciding the BIST session TL. As the HFS process runs fast, several experiments, with different test lengths, can be performed, providing the FC vs. the number of test vectors curve.

Figure 5 shows a typical FC(N) curve obtained using the new HFS tool. In order to obtain these curves, instead of using MISRs, the circuit is duplicated in the FPGA (one circuit for fault injection and one "golden" circuit) allowing fault dropping and identification of the first vector that detects each fault. Figure 5 presents the curve obtained exercising the c7552 with a LFSR implementing a polynomial of degree 168 and with taps in positions 166, 153 and 151. The seed used was 111…1 and the fault simulation time was 105 seconds.

6. Conclusions

BIST TE evaluation for stand-alone or embedded FPGA cores is a costly process. In this work, a novel HFS technique is proposed, that efficiently, using partial reconfiguration, ascertain (or not) *the BIST to be used during lifetime testing of these components.* The proposed HFS approach has been compared to a commercial software fault simulation (SFS) tool and the superiority has been demonstrated on two ISCAS'89 benchmark circuits. The new HFS tool simulates a clone from the ARM processor with 70576 faults and one million test vectors in two hours. The LUT extraction and partial reconfiguration processes were detailed.

A new, efficient and flexible LUT fault model was implemented. The faulty configurations that correspond to each fault are pre-computed only once, prior to fault simulation, and loaded in the beginning of HFS. This approach is more efficient and adds flexibility to the fault model.
The CLB fault list was detailed and the use of the LSA fault model at LUTs terminals was analysed as a fault sampling. Starting from a pessimistic 17% fault sampling if all CLB resources could be in use, we have shown that, due to the relevance of the LUTs on the implementation of the FPGA logic, the fault sampling is over 70% in practical cases.

Acknowledgement

This work has been partially funded by FCT (Portugal), under the project POCTI/41788/ESE/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] Ch. E. Stroud, “A Designer’s Guide to Built-In Self Test”, Kluwer Academic Pubs., ISBN 1-4020-7050-0, 2002.
[3] 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.
[4] 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.
[5] 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.
[6] 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.
[7] 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.
[8] Shih-Arn Hwang, Jin-Hua Hong and Cheng-Wen Wu, “Sequential Circuit Fault Simulation Using Logic Emulation”, IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems, vol. 17, no. 8, pp. 724-736, August 1998.
[9] P.Civera, L.Macchiarulo, M.Rebaudengo, M.Reorda, M.Violante, “An FPGA-based approach for speeding-up Fault Injection campaigns on safety-critical circuits”, IEEE Journal of Electronic Testing Theory and Applications, vol. 18, no.3, pp. 261-271, June 2002.
[10] M.B. Santos, J. Braga, I. M. Teixeira, J. P. Teixeira, "Dynamic Fault Injection Optimization for FPGA-Based Hardware Fault Simulation", Proc. of the Design and Diagnostics of Electronic Circuits and Systems Workshop (DDECS), pp. 370-373, April, 2002.
[11] M. Abramovici, P. Menon, “Fault Simulation on Reconfigurable Hardware”, *IEEE Symposium on FPGAs for Custom Computing Machines*, pp. 182-190, 1997.
[12] M. Abramovici, P. R. Menon, D. T. Miller, “Critical Path Tracing: An Alternative to Fault Simulation”, IEEE Design Automation Conference, pp. 468 - 474 , 1984.
[13] L. Burgun, F. Reblewski, G. Fenelon, J. Barbier, O. Lepape, “Serial fault simulation”, Proc. Design Auomation Conference, pp. 801-806, 1996.
[14] 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.
[15] 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.
[16] 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.
[17] 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 Conf. (MAPLD), pp. 27, 1999.
[18] E. Lechner, S. Guccione, “The Java Environment for Reconfigurable Computing”, Proc. of the 7th Int. Workshop on Field-Programmable Logic and Applications (FPL), Lecture Notes in Comp. Science 1304, pp.284-293, Sep. 1997.
[19] 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.
[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] A. Parreira, J. P. Teixeira,, M.B. Santos, "A Novel Approach to FPGA-based Hardware Fault Modeling and Simulation", Proc. of the Design and Diagnostics of Electronic Circuits and Syst. Workshop, pp. 17-24, April, 2003.
[23] Sheng Yu Shen, “nnARM core a ARM-7 clone”, http://www.opencores.org
[24] F. Brglez, D. Bryan, K. Kominski, “Combinational Profiles of Sequential Benchmark Circuits”, Proc. Int. Symp. on Circuits and Systems (ISCAS), pp. 1229-34, 1989.
[25] M.G. McNamer, S.C. Roy, H.T. Nagle, “Statistica Fault Sampling”, IEEE Trans. on Industrial Electronics, vol. 36, nº. 2, pp. 141-150, May 1989.