22nd IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems

Optimization of Self Checking FIR filters by means of Fault Injection Analysis

S. Pontarelli†, L. Sterpone‡, G.C. Cardarilli†, M. Re†, M. Sonza Reorda‡, A. Salsano†, M. Violante ‡

†Department of Electronic Engineering
University of Rome "Tor Vergata"
Rome, ITALY

‡Department of Electronic Engineering
Politecnico di Torino
Turin, ITALY

Abstract

In this paper the design of a FIR filter with self checking capabilities based on the residue checking is analyzed. Usually the set of residues used to check the consistency of the results of the FIR filter are based of theoretic considerations about the dynamic range available with a chosen set of residues, the arithmetic characteristics of the errors caused by a fault and on the characteristic of the filter implementation. This analysis is often difficult to perform and, to obtain an acceptable fault coverage the set of chosen residues is overestimated. obtained result a and therefore requires that Instead, in this paper we show how using an exhaustive fault injection campaigns allows to efficiently select the best set of residues. Experimental results coming from fault injection campaigns on a 16 taps FIR filter demonstrated that by observing the occurred errors and the detection modules corresponding to different residue has been possible to reduce the number of detection module, while paying a small reduction of the percentage of SEUs that can be detected.

I. Introduction

The silicon integrated circuits trend is characterized by a steady reduction in the feature size combined with a steady rise in density and speed [1]. A lot of new problems related to this incredible increase of complexity must be faced both from the technological and the architectural point of view. In this scenario, both permanent and transient fault probability increases limiting the silicon foundry yield and the reliability and availability of the implemented circuits. On the other hand, this trend allows to use digital systems to perform very complex Digital Signal Processing (DSP) operations, where the requirements in terms of speed and circuit complexity are very high, and the mandatory use of technologies with the best available feature size implies the increase of the probability of fault occurrence. Many works have been published on the detection of faults in arithmetic structures. In particular self-checking adders based on residue codes [2], [3], parity codes [4], or Berger codes [5] have been proposed. For FIR filters the RRNS (Redundant Residue Number System) [6], [7], [8] representation allow to easily detect a fault inside the filter. However, this approach have a serious drawback since it requires a certain area overhead for the forward and reverse conversion between binary and RNS representation and, moreover, the self-checking properties must be guaranteed also for these conversion circuits. The use of signed digit representation to detect fault in FIR filters is also possible [9]. In this work the self-checking property of the FIR filter is obtained by using technique based on residue checking. The detection of a SEU fault in an arithmetic block is performed by using some completely independent circuits that uses, as check symbols, the residues of the operands modulo a suitable base. The choice of the best set of residues is performed by means of a fault injection campaign. When the analysis of SEUs is the major concern, simulation-based fault injection approaches allow an early evaluation of the system dependability when only the model is available. However, considering the large complexity of such circuitry, a huge amount of CPU time may be required, thus limiting the number of faults that can be considered. Thus simulation based approaches are not a suitable solution to exhaustive evaluate the fault tolerance capabilities of Self-Checking circuits.

The usage of low cost commercial-off-the-shelf (COTS) FPGA devices for efficiently emulating a model of the system under analysis has been explored in several works [10] [11].

In [12] the extension to the injection of transient fault is proposed, where an operating model of the system under analysis is exploited. Although very efficient in reducing the CPU time needed for evaluating high numbers of faults, it mandates the insertion of fault-injection-oriented features that modify the structure of the model and therefore it cannot be exploited for evaluating self-checker circuitry.

The authors of [13] and [14] proposed alternative approaches to inject faults while emulating the system using FPGA devices, where partial reconfiguration is employed to perform the injection of SEUs. The most important benefit stemming from these approaches is that the source model of the system under analysis is not needed, while only a net-list suitable for being placed in an FPGA is required. On the one hand, the intellectual property of the IP core is preserved; on the other hand, the SEE analysis is performed on a model very similar to the one that will be employed in the final system, while in [12] the model must be changed to insert the fault-injection-oriented features. The major drawback of [13] is the speed: being based on JBits

1550-5774/07 $25.00 © 2007 IEEE
DOI 10.1109/DFT.2007.23

[16] and on a slow communication interface between the board carrying the emulated system, and the host computer managing the experiment, the number of faults that may be injected is quite limited. The authors reported that about 100 msec are needed for injecting, and classifying the effect of one SEU.

The major drawback of [14] is the portability: a specific custom-developed board is needed to perform fault injection. The system is very time-efficient (44 msec are needed for injecting and classifying the effect of one SEU), but it may be quite expensive to implement.

In this paper, we used a fault injection technique able to perform injection campaigns in a fraction of the time that simulation-based approaches require, thus supporting the execution of exhaustive fault injection campaigns. Furthermore, we used low-cost commercial-of-the-shelf FPGA devices for efficiently emulating a model of the system under analysis.

The main contribution of our work consists in demonstrating that the usage of exhaustive fault injection analysis in conjunction with the property of circuit's modules, allows to improve the efficiency of the checker. This approach has been previously used in [15] to optimize the design of a self-checking Reed Solomon decoder. In this paper we extend this approach to more common applications related to the processing of multimedia data. The example taken into account is a Finite Impulse Response (FIR) filter, but the combined approach of error detection using a residue checker and the optimization by means of fault injection campaign can be easily applied to most of the applications related to the Digital Signal Processing. In fact, all these applications use the basic arithmetic operations (addition and multiplication) used in the FIR filter and therefore can be checked using the residue checking approach originally presented in [2]. The best set of residue assuring a high fault detection coverage can be selected using the approach presented here.

The paper is organized as follows: in Section II a description of the fault injection environment is given. In section III the checker modules based on the computing of the residues used to detect faults inside the filter are described, while section IV describe the set-up and the results of the fault injection campaign and shows how these results are useful to improve the efficiency of the proposed self-checking scheme. Finally, the conclusions are presented in section V.

II. Fault injection Environment

The fault injection system we developed is composed of the following modules: a *host computer*; an *FPGA board* equipped with a Virtex II-Pro device, and a serial communication link to the host computer. The host computer is preliminary used for configuring the Virtex-II Pro and for the generation of a *fault location list*. During the execution of the fault injection experiment it only provides an user-friendly interface to run the fault-injection experiments and to collect the results in terms of fault-effect classification.

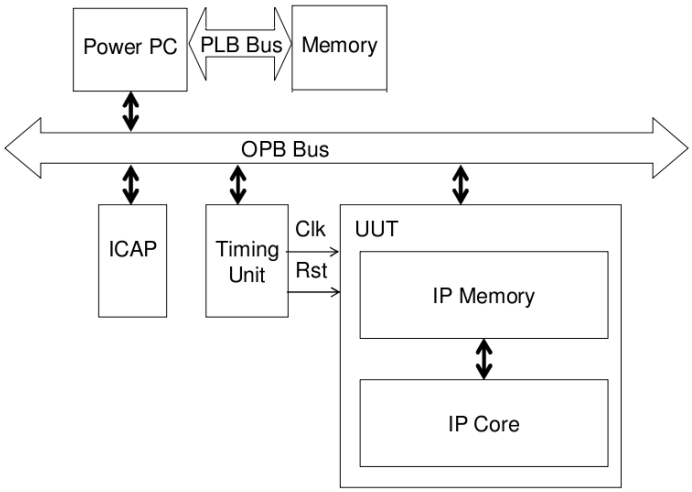

Fig. 1. Architectural scheme of the proposed fault injection approach

The FPGA board is the core of the fault-injection system and its layout is depicted in Fig. 1. It is composed of four components interconnected by an On-chip Peripheral Bus (OPB):

- *Timing Unit*: it drives the UUT clock and reset. The clock of the UUT has the same frequency of the FPGA device layout. A port connected to the OPB Bus defines its functionality.
- *Unit Under Test (UUT)*: it is the circuit under test and it may consist of an IP core and an own memory. The input and output ports of the IP core are connected to the OPB Bus while the reset and clock signals are connected to the Timing Unit.

- *ICAP*: it is the Internal Configuration Access Port provided by last generations of Xilinx FPGAs. It allows the access to the FPGA configuration memory through an internal port in order to perform partial reconfiguration without the support of an external hardware. For the purpose of this work we configured the ICAP in such way that it is able to access to all the memory elements (such Flip-Flops or Latches) of the UUT IP core.
- *PowerPC microprocessor*: it is hardwired in the FPGA device and it has two functionalities: At first, it performs the fault injection of SEUs within the memory elements of the UUT IP core, and it communicates the fault-injection experiment results to the host computer through a serial communication link.

The serial communication link is supported by a RS-232 cable that connects the FPGA board to the host computer.

The fault injection execution flow is a two-phase process composed of a preliminary phase followed by an execution phase. The flow of the preliminary phase is illustrated in Fig. 2. This phase is automatically executed by the host computer using either internal developed tools and commercial tools provided by Xilinx. The preliminary phase is executed following these actions:

- The Unit Under Test is inserted within the FPGA device layout circuit description. This actions is performed by the *UUT Wrapper Inserting Tool* that identifies the UUT input and output ports as well as the clock and reset signals. It generates an UUT wrapper that is inserted within the FPGA device layout description linking the input and output ports with the OPB Bus and the clock and reset signals to the Timing Unit.
- The FPGA device layout is compiled following the FPGA implementation tools chain provided by Xilinx. This action returns a Native Circuit Description (NCD) file that contains all the information about the configuration of the FPGA device.
- The BITGEN tool provided by Xilinx is then used to convert the obtained NCD file into a bitstream that is loaded within the FPGA configuration memory. Besides, BITGEN is used to generate a Logic Allocation file that contains a list of all the logic resources used within the FPGA.
- The *Fault Location List Generator* reads the Logic Allocation file identifying all the logic resources belonging to the UUT IP core and generates a Fault Location List file where each fault location is characterized by an identifier of the corresponding position of the flip-flop or latch within the FPGA.

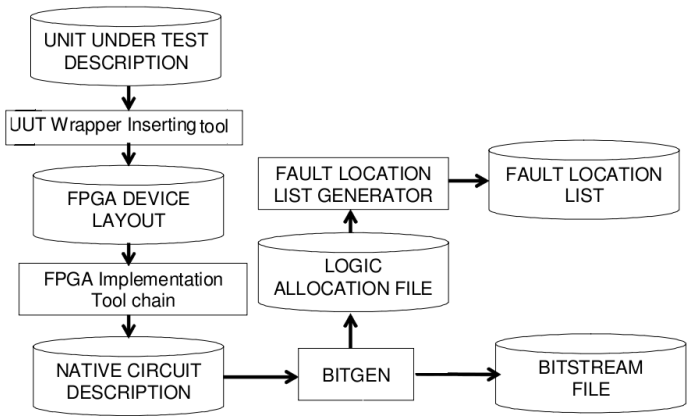

Fig. 2. Preliminary phase of the fault injection execution flow

The execution phase of the used fault injection approach consists of several procedures included within a fault-injector algorithm. The algorithm is executed by the PowerPC and it consists of three parts: pre-running, campaign and fault injection results. The Pre-running starts the fault injection experiment. At first, it loads within the PowerPC memory the test patterns that will be applied to the UUT and initializes the UUT IP memory (i.e. if the IP core is a processor the UUT IP memory will be loaded with the desired program). Secondly, it performs a golden run of the UUT storing the total number of Clock Cycle (CC) and the Golden Output (GO) produced. The Campaign performs the fault injection of the selected number of faults (NF). The following steps are executed for the injection of each SEU:

1) The UUT is resetted and the Timing is configured in such a way that it sends a reset to the UUT.
2) A fault injection time (FT) and a fault location (FL) are randomly selected considering the number of clock cycle CC and the set of FL available.
3) The execution of the UUT is started until it reaches the clock cycle FT. This operation is performed by configuring a Timing Unit's terminal counter at the FT value.

4) The fault location FL is read. This procedure reads directly the content of the flip-flop or of the latch from the configuration memory through the usage of the ICAP port.
5) The bitstream of the FPGA is partially reconfigured writing the opposite value within the content of the flip-flop or latches identified by FL. Therefore a SEU is injected in the considered fault location.
6) The execution of the UUT is continued until CC is reached. During the execution, it monitors the UUT output ports reading the data on the RS-232 interface and comparing their value with the UUT golden outputs. It finally updates a fault classification list (FCL) with the results obtained by the fault injection and classifying each injected SEU as silent, if the output produced by the UUT are equal to the GO; or wrong answer, if a mismatch was detected.

III. Detection of faults in the FIR filter

In this section the implementation of the FIR filter and of the blocks performing the modulo based error checking are described. The equation of a FIR filter is:

$$y(n) = \sum_{k=0}^{P} a_k \cdot x(n-k) \tag{1}$$

where $x(n)$ and $y(n)$ are the input and output sequences, and a_k are the filter coefficients. If no truncations are performed in the hardware implementation of the FIR filter the error detection can be done computing the residue modulo m of $y(n)$ directly starting from the residue modulo m of the input sequence $x(n)$. The result obtained by this computation can be seen as a check symbol to be used to detect fault inside the filter. In Fig. 3 an implementation of this basic scheme is presented, where the operations performed by the block computing $< y(n) >_m$ are performed modulo m. The block named equality checker can be implemented as a two-rail equality checker [17], [18] in order to avoid undetectable fault inside this block.

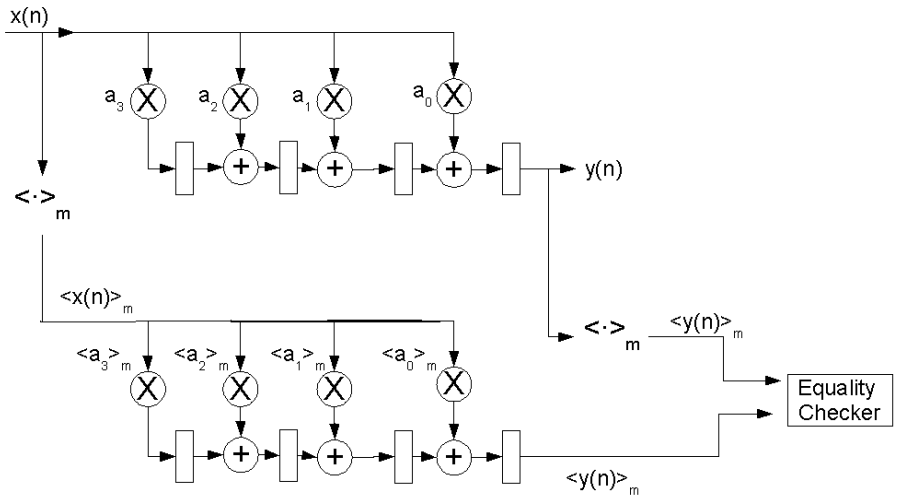

Fig. 3. Error detection scheme for a FIR filter

Considering this structure, the check symbol is able to detect a fault inside the filter only if the erroneous value caused by the fault is characterized by a check symbol that is different from the ones of the fault-free operation. This condition is verified when no aliasing between the modulo m of an erroneous result and the correct one occurs. Starting from the equation (1) we can write

$$< y >_m \neq < y + e >_m \Rightarrow < e >_m \neq 0 \Rightarrow e \ \neq k \cdot m \tag{2}$$

Where $< \cdot >_m$ is the modulo m operation, y in the output of the filter and $e \neq 0$ is the error generated by a fault in the filter. The last implication means that the error is not a multiple of the chosen modulo base m. The probability that the no aliasing condition is respected depends by different factors such as the value of the filter coefficients, the filter implementation and the assumed fault set. The probability that no aliasing occurs can be increased by using a set of different moduli $m_1, m_2 \dots m_n$ and comparing in parallel the different check symbols obtained. It must be noticed that the no aliasing condition is verified for almost one of the modules (and therefore the fault can be detected) if and only if the same condition is verified for a module $M = lcm(m_1, m_2 \dots m_n)$, where lcm is the least common multiplier of the set of moduli. The proof of this claim can be

easily done using the Chinese Remainder Theorem. A overestimation of the moduli set that assure the detection of all the faults inside the filter is given by a set of moduli $m_1, m_2 \dots m_n$ which have as lcm a number greater than the dynamic range of the filter output. In fact, if we use N bits for the signed binary representation of the y output the error is constrained to be in the range $-2^{N-1} \leq e < 2^{N-1}$ and the no aliasing condition is satisfied if M is greater than 2^N. In section IV is shown how the fault injection campaign allows to reduce the magnitude of M (and therefore the set of modules $m_1, m_2 \dots m_n$) paying a small reduction of the percentage of SEUs that can be detected.

IV. Experimental Setup and Fault Injection Results

In order to perform the fault injection campaign on a 16 taps FIR filter with coefficient and input length of 8 bits and output length of 20 bits, the hardware blocks that has been designed and implemented on the Xilinx FPGA are:

1) The 16 taps FIR filter,
2) a Linear Feedback Shifter Registers (LFSR) that act as pseudo-random number generators, providing the input data x for the filter,
3) a set of blocks providing the input modulo reduction $< x >_m$ for the set of bases used for the analysis,
4) a set of modulo m_i FIR filters in which the addition and multiplication operation are performed modulo m_i.
5) a set of blocks providing the filter output modulo reduction $< y >_m$ for the set of residues used for the analysis
6) a set of blocks comparing the results of the output $< y >_m$ with the corresponding output of the FIR filter that perform the modulo m_i operation. These blocks provide as outputs the signals $out_1 \dots out_n$ signaling if an error has been detected by the corresponding m_i module.

The set of moduli used to perform the analysis is 3,5,7,9,11,13,15,31 with $lcm(3, 5, 7, 9, 11, 13, 15, 31) > 2^{20}$. The use of the complete set is therefore able to detect all the faults inside the filter. Fig.4 shows these blocks and their interconnections.

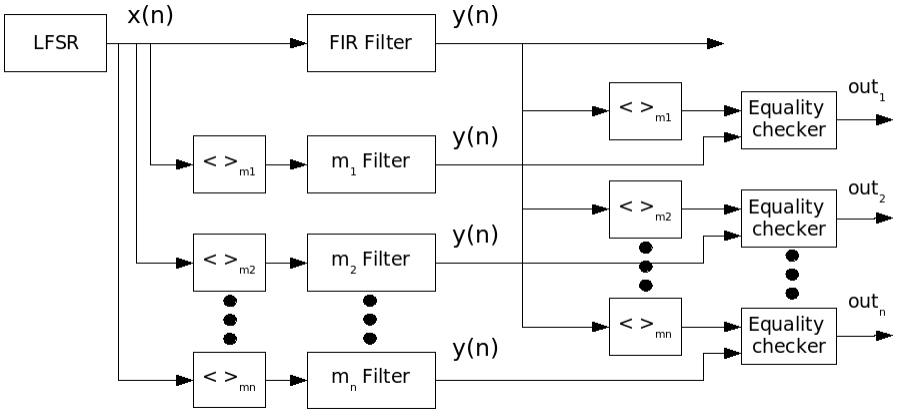

Fig. 4. Set-Up for fault injection

The experimental campaigns were based on evaluating the self checking FIR filter dependability and observing the fault detection capability of the adopted self-checker. For these purposes, we performed a fault injection campaign able to inject in a random fault injection time and in a random fault location of the filter 124,289 SEUs, corresponding to 100,000 activated faults. During the execution of the fault injection experiment we fixed the total number of clock cycle (CC) to 20,000 and the fault injection time (FT) at a random clock cycle between 0 and 19,000. The outputs $out_1 \dots out_n$ has been monitored and we assume that the fault is activated if at least one of the outputs of the UUT differs by the corresponding output obtained during the golden run and therefore we associate the fault detection with the corresponding module.

The results achieved from the fault injection campaign report that for the 100,000 activated faults the complete set of moduli detects 99993 faults, corresponding to a fault coverage up to 99,99%. We examined the output of each detection module obtaining the results shown in table I,where in the first column is reported the name of the module corresponding to the monitored output.

As the reader can observe, the element performing the operation modulo 15 is able to detect almost all the fault injected in the FIR filter. Therefore the designer can use only this module avoiding the use of the other modules reducing the overhead of the filter and paying only a small reduction in terms of fault detection coverage. To improve the fault detection coverage we also compute the percentage of faults detected by some subset of modules. In particular we focus the attention on the modules that yet alone have an high percentage, namely 15,11,9,7. The fault detected by the couples $15, 11$,$15, 9$, $15, 13$ and $15, 7$ and by the triples $15, 13, 11$, $15, 11, 9$, $15, 11, 7$ are reported in table II.

Module	Detected SEUs	Detected SEUs [%]
M_3	3874	3
M_5	47334	47
M_7	91574	91
S_9	95083	95
S_{11}	97416	97
S_{13}	87908	87
S_{15}	99632	99
S_{31}	2895	2

TABLE I

FAULT COVERAGE OF A SINGLE MODULE

Module	Detected SEUs	Detected SEUs [%]
$15, 11$	99991	99.99
$15, 9$	99979	99.97
$15, 13$	99944	99.94
$15, 7$	99966	99.96
$15, 11, 13$	99993	99.99
$15, 11, 9$	99993	99.99
$15, 11, 7$	99993	99.99

TABLE II

FAULT COVERAGE OF MODULES PARTITION

From this table we can see that using one of the subsets $15, 13, 11$, $15, 11, 9$, $15, 11, 7$ we can obtain the maximum fault detection coverage, while using only the two modules $15, 11$ we detect obtain 99991 faults. These results allow to identify as completely unnecessary some modules, namely the modules 3, 5, 31, while only a subset of the other modules is necessary to obtain the maximum fault detection coverage.

V. CONCLUSIONS

The results presented in this paper shows how a fault injection analysis of a complex structure allows to identify which kind of errors are more probable to occurs due to a fault inside the circuit under test. This analysis is very useful for designing the additional modules that allows to implement a self-checking version of the circuit under analysis. In particular, for structures with well defined arithmetic properties, such as the FIR filters, this campaign allows to identify which blocks must be added avoiding the overestimation of the possible fault effects.The results presented shows that some modules can be excluded from the self-checking scheme with a very little reduction of the fault tolerance capabilities. An extensive fault injection campaign in a fast and flexible environment is necessary to improve the efficiency of self-checking structure. The fault injection technique used in this paper allows to perform exhaustive injection campaigns on very complex structure in a incomparable less time than simulation-based approaches. Moreover, it uses a low-cost commercial-of-the-shelf FPGA devices and do not requires modification in the internal structure of the circuit under test. This a mandatory characteristic because the proposed use of the fault injection analysis and the obtained optimization are very dependent on the internal structure of the circuit.

REFERENCES

[1] "2001 International Technology Roadmap for Semiconductors", http://public.itrs.net/.

[2] W. W. Peterson, "On checking an adder", I.B.M. J. Res. Develop., vol 2, pp. 166-168, Apr 1958.

[3] D. Nikolos, A.M. Paschalis, G. Philokyprou, "Efficient design of totally self-checking checkers for all low-cost arithmetic codes", IEEE Trans. on Computers, vol. 37, N. 7, pp. 807 - 814, July 1988.

[4] M. Nicolaidis, "Carry checking/parity prediction adders and ALUs", IEEE Transactions on very Large Scale Integration (VLSI) Systems, Vol. 11, N. 1 , Feb 2003 pp. 121-128.

[5] J.-C. Lo, S. Thanawastien, T. R. N. Rao, M. Nicolaidis, "An SFS berger check prediction ALU and its application to self-checking processors design", IEEE Trans Computer-Aided Design, pp. 525-540, Mar. 1992.

[6] S. Bandyopadhyay, G.A. Jullien, A. Sengupta, "A systolic array for fault tolerant digital signal processing using a residue number system approach", Systolic Arrays, 1988., Proceedings of the International Conference on 25-27 May 1988 Page(s):577 - 586

[7] L. Imbert, G.A. Jullien, V.S. Dimitrov, A. Garg, "Fault tolerant complex FIR filter architectures using a redundant MRRNS", Signals, Systems and Computers, 2001. Conference Record of the Thirty-Fifth Asilomar Conference on Volume 2, 4-7 Nov. 2001 Page(s):1222 - 1226 vol.2

[8] W. K. Jenkins, "The Design of Error Checkers for Self-Checking Residue Number Arithmetic", Computers, IEEE Transactions on Volume C-32, Issue 4, Apr 1983 Page(s):388 - 396

[9] G. C. Cardarilli, S. Pontarelli, M. Re, A. Salsano, "Fault tolerant design of signed digit based FIR filters" Circuits and Systems, 2006. ISCAS 2006. Proceedings. 2006 IEEE International Symposium on 21-24 May 2006

[10] S. A. Hwang, J.H. Hong, C.W. Wu, "Sequential circuit fault simulation using logic emulation", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 17, No. 8, Aug. 1998, pp. 724-736

[11] K.T. Cheng, S.Y.Huang, W.J. Dai, "Fault Emulation: A new methodology for fault grading", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 18, No. 10, October 1999, pp. 1487-1495

[12] P. Civera, L. Macchiarulo, M. Rebaudengo, M. Sonza Reorda, M. Violante, “Exploiting Circuit Emulation for Fast Hardness Evaluation”, IEEE Transactions on Nuclear Science, Vol. 48, No. 6, December 2001, pp. 2210-2216
[13] L. Antoni, R. Leveugle, B. Feher, “Using Run-time Reconfiguration for Fault Injection in Hardware Prototypes”, IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, 2000, pp. 405 - 413
[14] J. Tombs, M.A. Aguirre, “FT-UNSHADES”, Microelectronics Presentation Day, 2004
[15] S.Pontarelli, L.Sterpone, G.C.Cardarilli, M.Re, M.Sonza Reorda, A.Salsano, M.Violante, “Self Checking Circuit Optimization by means of Fault Injection Analysis: A Case Study on Reed Solomon Decoders”, 13th IEEE International On-Line Testing Symposium, Crete, Greece, July 2007
[16] JBits 2.8, Xilinx, San Jose, CA, 2001
[17] P. K. Lala, ”Self-Checking and Fault-Tolerant Digital Design”, San Mateo, CA: Morgan Kaufman, 2001.
[18] D. K. Pradhan, ”Fault Tolerant Computing, Theory and Techniques”, Edited by Prentice-Hall, 1986.