Performance Analysis and Improvements for a Simulation-based Fault Injection Platform

O. Ruano, J.A. Maestro, P. Reviriego
Dept. Ingeniería Informática -Universidad Antonio de Nebrija
Madrid, Spain
Email: {oruano, jmaestro, previrie}@nebrija.es

Abstract—**In this paper, we study and present two techniques to improve the performance of a simulation-based fault injection platform that inserts bit flips in order to model soft errors on digital circuits. The platform is based on the ESA Data Systems Division's SEE Simulation Tool. In contrast with methods based on emulation, the proposed approach reduces the complexity and costs, supplying a test environment with the same reliability as emulation systems. Only one disadvantage appears when comparing both methodologies: the lower performance of the simulation in cases where the fault injection campaigns are very large. Two proposals have been developed in order to address this drawback: the first one is based on software (through checkpoints) and the second one uses parallel computation.**

I. INTRODUCTION

Microelectronic circuits that have to operate in harsh environments, like space or nuclear reactors, are exposed to the radiation effects which produce certain risks [1]. One type of hazard is Single Event Effects (SEEs) that cause changes in the values of flips-flops (SEUs) or combinational logic (SETs). In order to guarantee the circuit reliability, designers of rad-hard systems have two choices: Radiation Hardening by Process (RHBP) and Radiation Hardening by Design (RHBD) [2]. The first option includes physical techniques that must change the manufacture process like shielding or silicon on insulator (SOI). One alternative with less cost is to use standard CMOS with no additional masks, providing hard memory elements like HIT cells [3]. Today's relatively small market for radtolerant components makes it difficult for the few remaining suppliers to offer economical solutions. This has prompted designers to use the second choice: radiation-hardening-by-design (RHBD) approach. In RHBD, electronic components are manufactured to meet specified radiation performance criteria, but the techniques employed to meet these criteria are implemented either in layout or in the application architecture and not in the fabrication process. This methodology applies, to make resilient designs, some well-known general hardening techniques like the use of redundancy [4] (Triple Modular Redundancy, TMR) or EDAC codes (parity, hamming). Another alternative consists in implementing specific techniques applying the "system knowledge" that increases the efficiency respect to the conventional techniques mentioned above [5]. To check those techniques, designers usually require mechanisms to simulate their implementations and study the effect of possible SEEs. Fault Injection is a feasible practice to achieve this purpose. Although there are several kinds of fault-injection, this paper will focus on the so-called software-based injection. Several examples presented in literature can be found in the related work section. Focusing on the simulation platform used in this paper (see Fig. 1) [6], the main advantages that can be highlighted are:

- Analysis in the firsts steps of the design cycle.
- Suitable when a prototype is not available.
- Capable of injecting faults into both synthesizable and non-synthesizable models.
- Good controllability: where and when a fault is injected.
- Observability: capable to monitor internal events.
- Easy and free distribution to the research community.

Conversely, the drawback is the high CPU time spent to simulate the model. This paper addresses some techniques to reduce the simulation time required by the platform to run the fault injection campaigns when studying the behaviour of the designs in the presence of SEEs.

The first exploits the simulator features and the second one implements a parallel system to perform the campaign computation. To assess the success of both techniques, we have performed fault injection experiments on an Intel i8051 microcontroller executing an elliptic filter.

The rest of the paper is organized as follow: related work is discussed in Section II. Section III describes briefly the architecture of the simulation system and its principles.

Section IV presents a performance analysis for several campaigns for an elliptic filter. Section V explains in depth the proposed

This work was supported by the Spanish Ministry of Education and Science under Grant ESP-2006-04163.

approaches. Finally, in Section VI conclusions and future work are presented.

II. Related work

Fault injection is a widely used technique for fault tolerance evaluation. The fault injection techniques presented in literature have been implemented using hardware as well as software techniques [7]-[14]. A hardware approach consists of injecting physical faults into the target system hardware. These methods have the advantage of causing errors which may be close to a realistic fault. However, these approaches require special hardware and the injector tools are dedicated to a specific target. The main problem is the complexity to control and observe the fault effects that usually requires an I/O comparison cycle by cycle between the tested system and a non exposed twin system. In [7][8], a framework under the name of FT-UNSHADES has been developed in collaboration with ESA Data Division System. It is composed of two Xilinx FPGAs where the main one is a Virtex II capable of holding two parallel implementations of design and the second one is a SPARTAN-3 which acts as fast bridge between the injector software and the Virtex. The main feature is readback mechanism and the partially read and written configuration memory.

In [9], an emulation-based platform has been developed, and the principle is similar to the FT-UNSHADES: internal partial reconfiguration feature. It is composed of an FPGA board equipped with a Virtex II-Pro device, and a serial communication link to the host computer that acts as injector of faults. This system takes advantage of the new internal Configuration Access Port provided by last generation of Xilinx FPGAs called ICAP, to improve the partial reconfiguration.

In [10], fault injection is carried out at the pin level to measure fault latency in multiprocessor systems. As a disadvantage, it requires a special hardware to inject faults. .

However, in [11], an environment for controlling the injection of faults in a distributed system is implemented, where their experiments show the feasibility of software techniques for the validation of systems.

In [12], a software fault injection and monitoring environment is presented on a parallel machine build around PARIX operating system called Xception.

In [13], a software tool is developed on a SPARC workstation for the validation of fault tolerant computer system under the name of FERRARI.

Finally, in [14], a fault injection environment, called HYBRID is presented. Faults are injected in memory via software and extra hardware is used to trace fault activation and propagation in the target system.

The technique proposed in the present paper is framed within the software-based simulation mechanisms where the design is simulated with a Hardware Description Language like VHDL or Verilog, and the upsets are injected through a test bench designed for this purpose via software.

III. Description of the simulation platform

The motivation behind the development of this platform is the need for a flexible and powerful fault injection system to help the designers to predict and explore potential weak points on ICs sensitive to hazards [5]. It can also be used to assess the effectiveness of a given protection technique [6].

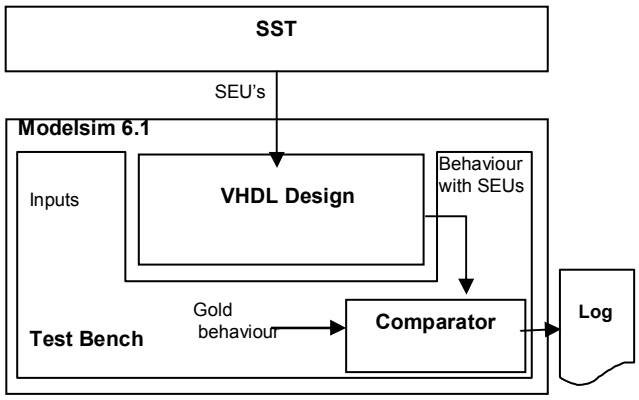

Fig. 1 Proposed Error Simulation Platform.

It works together with a commercial simulator (ModelSim version 6.3) using the built commands and capabilities of the simulator engine. In addition to the advantages of the simulation injection systems showed in the introduction this method minimizes the intrusiveness

IV. Performance analysis

A first experiment has been conducted in order to prove the low performance associated to the platform when no improvements are applied. The i8051 has been used through a VHDL description at the register-transfer level (RTL). We have considered a first fault model composed by a single bit-flip per simulation (99999 clock cycles and a 100-MHz frequency), running on one PC with an Intel(R) Pentium(R) M processor at 2.00 GHz and a 1Gb RAM. This fault model has been inserted in both the RAM and ROM memories of the processor.

TABLE I: EXPERIMENTAL RESULTS OR THE SST SIMULATION PERFORMANCE (IN SECONDS)

	Campaign Size	TIME [sec]
BENCHMARK Elliptic Filter	10 SEUs	2.73
	200 SEUs	53.26
	1,000 SEUs	265.29
	2,000 SEUs	531.51
	4,000 SEUs	1063.43
	7,000 SEUs	1866.20
	10,000 SEUs	2672.70
	100,000 SEUs	26629.62

Several tests have been carried out experimentally applying different campaigns composed of several simulations, and therefore several SEUs. The CPU times spent by the platform are shown in Table I. It is interesting to analyze the behavior of the platform in extreme conditions (e.g. the 100000-SEU campaign) where 7,4 hours were needed to perform the injection. This proves the low performance of the platform under these circumstances, especially noticeable when compared with HW-based systems, like the one in [15].

In order to mitigate this weakness (common to any simulation-based fault injection system), two enhancements will be presented and studied both in an analytical and experimental way: i) the use of checkpoints (a usual feature in VHDL simulators) and ii) parallelism as a technique to improve computation.

V. PROPOSED APPROACHES

In order to address the performance issues common to any simulation-based fault injection system, two techniques are proposed:

- The first technique exploits checkpoint features supplied by modern VHDL simulators.
- The second technique uses several CPUs in order to speed-up simulations.

A. First optimization: Checkpoints

This section presents an optimization to improve the simulation time of the platform based on the simulator checkpoint feature. In this way, we take advantage of the possibility to store the simulation state at a certain time, and then restore it later. This means that if a SEU needs to be injected at the end of a long simulation (of T_{sim} seconds, see Fig. 2), a first execution will happen at the beginning (up to the injection point), and then the simulation state would be saved.

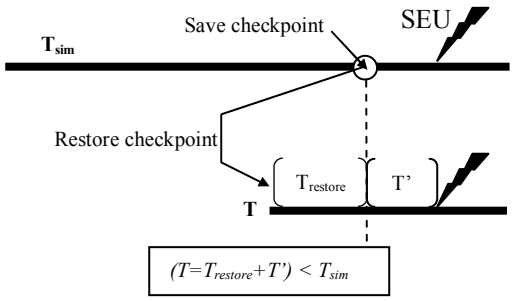

Figure 2: Checkpoint Optimization.

Later, if several experiments have to be conducted, that state could be restored (consuming $T_{restore}$ seconds), and only the final parts of the simulations would have to be run (needing only T' seconds). Obviously, in order to make this technique efficient, the following inequality must be met:

$$T_{restore} + T' < T_{sim} \quad (1)$$

In order to evaluate the effectiveness of this approach, we have carried out an experiment on the elliptic filter benchmark. For this experiment, the specific case of 10000 SEUs has been considered, with a simulation campaign of 99999 cycles and a 100-MHz frequency, running on the same PC described before. The first column of Table II shows the CPU simulation time for the 10000 SEUs without any optimization. The second and third columns illustrate the new times obtained with the added checkpoint optimization. Both differ in the time where the checkpoint was inserted: in the first case ($T_1=T_{restore1}+T_1'$) the checkpoint is located at 3/4 of the simulation and in the second case ($T_2=T_{restore2}+T_2'$), almost at the end of the simulation (in both cases the SEUs were inserted at the end of the simulation). The next columns show the achieved performance speedup ($SU_n = T_{ini}/T_n$) and the percentage of improvement produced by that speedup.

TABLE II: CHECKPOINT OPTIMIZATION FOR 10000 SEUS

Benchmark	Tsim [sec]	T1 [sec]	T2 [sec]	SU1	SU2
Elliptic Filter	2672.71	2490.93	1994.86	1.072	1.347
Improvement [%]				7.2%	34.7%

The results obtained in Table II are promising and show a significant improvement in performance. An improvement of 7.2% has been obtained in the less favourable case, SU1. In the best case, SU2, the improvement has increased to 34.7%. In

summary, the most beneficial cases happen when the checkpoint is as close as possible to the SEU appearance. However, if the SEU appears early enough in the simulation, the $T_{restore}$ penalty when using the checkpoint can make the overall time worse. Therefore, a detailed study should be made for each particular fault injection campaign, in order to find out where and how many checkpoints need to be added to get an optimal performance. The results obtained with this technique are depicted in Fig. 3.

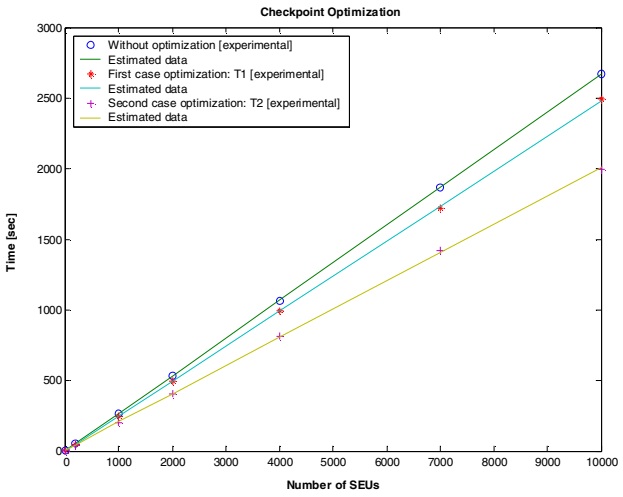

Fig. 3: Checkpoint results.

B. Second optimization: Parallelism

This section presents an optimization oriented to large fault injection campaigns based on parallelism. Parallel simulation is used for tasks which require very large amounts of computation and has been traditionally motivated by numerical simulations of complex systems such as weather and climate, chemical and nuclear reactions, biological (human genome), electronic circuits [16], etc. In our case, it has been motivated by the large computational time needed when the fault injection campaigns involve a large number of fault injections on a complex circuit. By dividing the injection campaign among a number of PCs, several benefits can be achieved:

- Save computation time.
- Being able to tackle large simulation campaigns.
- Cost savings, taking advantage of non-local resources using available computers in a network.

The most appropriate model for this problem is a distributed system architecture. The initial simulation campaign is divided into smaller parts by a central node (dispatcher), and then distributed to the rest of computers, which perform the simulation and return the results back to the central processor, through a message-passing model. This type of problem is called embarrassingly parallel because very little task interaction is required, and therefore the time overhead introduced by parallelism is limited, making the model purely scalable.

Based on this architecture, our purpose is to evaluate the performance of the analytical model and validate it through some fault injection experiments made on a network built on a star topology. In parallelism, the use of the highest number of processors in order to increase the parallel speedup may not be feasible due to the overhead introduced by communications, as mentioned before. In other words, adding an extra processor to a system with n nodes may not result in a performance increase, since the communication time generated by this extra node with the rest of the system may be larger than the computation time saved by introducing it. This is the reason why it is so important to study the scalability of the system, or on technical words, the isoefficiency. This refers to a parallel system ability to proportionately increase the parallel speedup when new processors are added. According to [17], the efficiency (E) of this type of systems must be kept constant, O(1), in order to be scalable:

$$E = \frac{T_S}{p.T_p} = \mathrm{O}(1) \tag{2}$$

where T_s is the simulation time in one processor and T_p is the time in the distributed system when it uses p processors. This implies that the work of solving the problem in a parallel system should have the same asymptotic growth (when the simulation size increases) as the sequential algorithm in a single processor:

$$O(T_s) = O(p.T_p) \tag{3}$$

This is what we need to prove in order to conclude that the provided solution is independent of (not limited by) the size of the problem (in number of SEUs).
Starting from a problem size equal to n (iterations-SEUs) and p processors, the parallel time will be

$$p \cdot T_p = T_s + T_o = T_s + p \cdot T_{comm} \tag{4}$$

$$T_p = \frac{T_s}{p} + \frac{T_o}{p} = \frac{T_s}{p} + T_{comm} \tag{5}$$

where T_s is the serial time (one processor), T_o is the total overhead time for a parallel system and T_{comm} is the communication time between two nodes:

$$T_{comm} = \frac{T_o}{p} \quad (6)$$

In the previous section, it has been experimentally proved that the serial time is a linear function (see Table I), so:

$$T_s = A + Bn \quad (7)$$

$$O(T_S) = O(n) \quad (8)$$

Now, we have to study the asymptotic growth of $p \cdot T_P$ (see (4)):

$$O(p \cdot T_p) = O(T_s + p \cdot T_{comm}) = \max(O(T_s), p \cdot O(T_{comm})) \quad (9)$$

The complexity of the first term is known, so the next step will be to experimentally study the magnitude of T_{comm} on the star topology network using a different number of machines (2, 3 and 4 machines respectively). In this case, the different machines in the network are Pentium 2.40 GHz with 256 Mb RAM.

The experimental results prove that the communication time between the dispatcher and any node is constant, regardless of the number of PCs and the size of the experiment (number of SEUs). This is reasonable since i) nodes do not interact among them (only with the dispatcher, and therefore increasing the number of nodes should not affect communications) and ii) the simulation script sent from the dispatcher to the nodes is independent of the number of SEUs that are injected. Therefore, it can be concluded that $O(T_{comm}) = O(1)$, i.e., it is constant.
Now, (9) can be rewritten as:

$$O(p \cdot T_p) = O(T_s + p \cdot T_{comm}) = \max(O(n), p \cdot O(1)) \quad (10)$$

And since n >> p always (more SEUs injected than PCs), then

$$O(p.T_p) = O(n) \quad (11)$$

Combining (8) and (11) it is clear that (3) is met, and therefore the system is isoefficient. This means that no matter how large the SEU campaign is, it will always be possible to add a new PC that will help to keep the simulation time feasible bounded, in spite of the communication overhead that this new PC may introduce. A different issue is the saturation level of the dispatcher, whose performance can decrease when a large number of nodes are added. However, since the interaction among nodes is so light, the number of processors to produce this problem is so large that should not be an issue for the proposed methodology.
In Table III, the simulation times obtained for different PCs and several SEU campaigns are depicted. It is seen that, in general, this simulation time decreases as more PCs are used, what is very convenient for large SEU campaigns. However, there is one exception: in the case of 10 SEUs, the results are worse when new PCs are added. This is reasonable, since in this case n (10 SEUs) is low enough so that the relation n >> p is not met. In the other cases, the results obtained show a significant improvement in performance. A sample of this could be the improvement obtained in the case of 10,000 SEUs, which is around 194% for 4 processors.

TABLE III: SIMULATION TIMES (IN SECONDS) FOR DIFFERENT NUMBER OF PCS AND SEUS

Benchmark Elliptic Filter	1 PC	2 PCs	3 PCs	4 PCs
10 SEUs	2.70	4.49	5.19	5.50
100 SEUs	27.33	18.43	15.70	13.99
200 SEUs	53.60	35.93	28.22	23.19
1,000 SEUs	266.45	162.98	132.34	99.10
10,000 SEUs	2689.35	1593.12	1275.37	915.01
100,000 SEUs	26637.25	16688.84	11463.37	9008.29

The results in Table III are graphically depicted in Fig. 4, where the time optimizations can be seen for the different number of nodes.

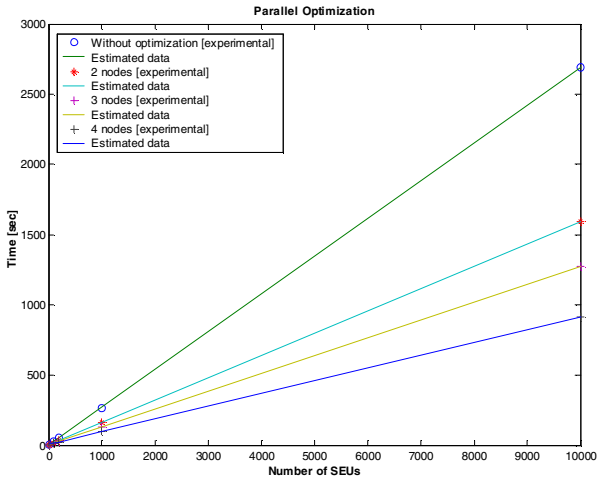

Fig. 4: Parallelism results.

VI. Conclusions and future work

In this paper, new proposals to improve the performance of the SST simulation-based fault injection platform have been presented. These new techniques have been developed using both the features supplied by the ModelSim 6.3 simulator and parallel computation through a distributed system. The benefits of applying these methodologies are clearly proved with the experimental results that have been obtained, increasing the platform performance, and therefore being able to face larger injections campaign. The use of this simulation environment that enables a flexible testing of the effects of SEUs will facilitate the future work, which will focus on the comparison between the SST injection results and real radiation tests, in order to measure the error rate prediction of the platform.

References

[1] P. Dodd and L. Massengill, "Basic Mechanisms and Modeling of Single Event Upsets in Digital Microelectronics", IEEE Transactions on Nuclear Sciences, Vol. 50, Issue 3, pp.: 583 – 602, June 2003.

[2] H.J. Barnaby, "Will radiation hardening by design work?", Nuclear & Plasma Sciences Society News, 2005.

[3] D. Bessot and R Velazco, "Design of SEU-Hardened CMOS memory cells: the HIT cell", IEEE RADECS, Vol. 13, Issue 16, pp.: 563 – 570, Sep 1993.

[4] W. Chen, R. Gong, K. Dai, F. Liu and Z. Wang, "Two New Space-Time Triple Modular Redundancy Techniques for Improving Fault Tolerance of Computer Systems", Computer and Information Technology, CIT, pp.: 175 – 175, Sept. 2006.

[5] P. Reyes, P. Reviriego, J.A. Maestro and O. Ruano, "New Protection Techniques against SEUs for Moving Average Filters in a Radiation Environment", IEEE Transactions on Nuclear Science, Vol. 54, Issue 4, pp.: 957 – 964, Aug. 2007.

[6] O. Ruano, J.A. Maestro, P. Reyes and P. Reviriego, "A Simulation Platform for the Study of Soft Errors on Signal Processing Circuits through Software Fault Injection", Proc. of IEEE International Symposium on Industrial Electronics, pp.: 3316-3321 June 2007.

[7] J. M. Tombs, M. A. Aguirre, F. Muñoz, V. Baena, A. J. Torralba, A. F. Leon and F. Tortosa, "An FPGA based hardware emulator for the insertion and analysis of Single Event Upsets in VLSI Designs", Proceedings of RADECS, pp.: 10000-10004, Sept. 2004.

[8] M. A. Aguirre, J. Noel, V. Baena , F. Muñoz , A. lbañez, A. Fernández and F. Tortosa: "Ft-Unshades: a New System for SEU Injection, Analysis and Diagnostics Over Post Synthesis Netlist", Mapld International Conference Washington D.C., USA NASA Office of Logic Design, 2005.

[9] L. Sterpone and M. Violante, "A New Partial Reconfiguration-based Fault-Injection System to Evaluate SEU Effects in SRAM-based FPGAs", IEEE Transactions on Nuclear Science, Vol. 54, Issue 4, pp.: 965 – 970, Aug. 2007.

[10] E. Jenn, J. Arlat, M. Rimén, J. Ohlsson and J. Karlsson, "Fault Injection into VHDL Models: The MEFISTO Tool", Proceedings of the 24th International Symposium on Fault Tolerant Computing, IEEE, Vol. 23, Issue 25, pp.: 168 – 173, Jun 1998.

[11] S. Han, K.G. Shin and H.A. Rosenberg, "Doctor: An Integrated Software Fault-Injection Environment for Distributed Real-Time Systems", Proc. IEEE Int. Computer Performance and Dependability Symposium, Vol. 24, Issue 26, pp.: 204 – 213, Apr 1995.

[12] J. Carreira, H. Madeira and J. Silva, "Xception: Software Fault Injection and Monitoring in Processor Functional Units", Conference on Dependable Computing for Critical Applications, 1995.

[13] G.A. Kanawati, N.A. Kanawati and J.A. Abraham, "FERRARI: A Flexible Software-Based Fault and Error Injection System", IEEE Transactions on Computers, Vol. 44, Issue 2, pp.: 248 – 260, Feb 1995.

[14] A. Ejlali, S.G. Miremadi, H. Zarandi, G. Asadi and S.B. Sarmadi, "A hybrid fault injection approach based on simulation and emulation co-operation", Dependable Systems and Networks, 2003.

[15] O. Ruano, P. Reyes, J.A. Maestro, L. Sterpone, and P. Reviriego, "An Experimental analysis of SEU Sensitiveness on System Knowledge-based Hardening Techniques", IEEE DDCESS, Vol. 11, Issue 13, pp.: 1 – 6, April 2007.

[16] G. Komar, "Compute Farm Solution for Electronic Design Automation", White paper, ZORAN Microelectronics.

[17] A. Grama, A. Gupta, G. Karypis and V. Kumar, "Introduction to Parallel Computing", Addison Wesley, 2003.