A Simulation Platform for the Study of Soft Errors on Signal Processing Circuits through Software Fault Injection

O. Ruano, J.A. Maestro, P. Reyes
Universidad Antonio de Nebrija
Madrid, Spain
Email: oruano@nebrija.es, jmaestro@nebrija.es, preyes@nebrija.es

P. Reviriego
Universidad Carlos III
Madrid, Spain
Email: revirieg@it.uc3m.es

***Abstract*— In this paper we present a simulation platform tailored for Signal Processing Circuits that injects bit flips in order to model soft errors. The platform is based on the ESA Data Systems Division's SEE Simulation Tool upgraded with new functionalities. In order to show the effectiveness of the platform, a digital filter has been tested.**

I. INTRODUCTION

The growing need in the aerospace industry to deal with space radiation environments in order to ensure the success of missions, has spurred several lines of research concerning the radiation effects on electronic systems [1]. These disturbances affect the behaviour of the devices, causing failures and soft errors [2]. For example, a well-known risk caused by the Van Allen Belts is the South Atlantic Anomaly, which affects many Low Earth Orbiting Satellites [3] (Fig. 1).

These kinds of hazards are known as Single Event Effects (SEE), and in particular, Single Event Upsets (SEUs) are a major concern when dealing with electronic designs that suffer the consequences of a radiation environment [4].

To address this issue, a new simulation platform is presented in this paper, which allows the injection of SEEs in circuits at a higher level of abstraction. This tool allows us to exploit the potential benefits that can be obtained by analyzing the effects of SEEs in the earlier stages of the design process. The results of this analysis could help in detecting weaknesses and in designing new SEE protection techniques.

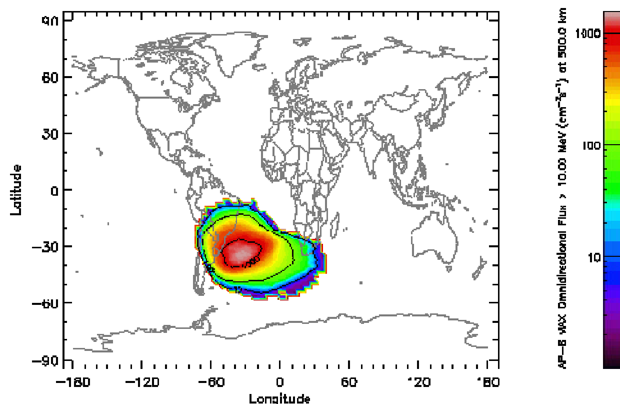


Fig. 1: South Atlantic Anomaly

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

The paper is organized as follows: in section 2, the related work carried out in this area is summarized. In section 3 we explain the fault injection platform that has been developed; and in section 4 a case study where the platform is used to compare the effects of SEUs on two different structures of a moving average filter is reported, showing the effectiveness of the platform in identifying weaknesses caused by SEUs. The case study is also used to assess the performance of the platform in terms of simulation time which is one of the major problems with software-based fault injection approaches.

II. RELATED WORK

A number of platforms have been developed to evaluate the effects of radiation on electronics circuits. They are used to assess the impact of SEEs on circuits during the design process or once the device has been manufactured. Such platforms can follow two different procedures:

- Real Radiation Tests on the Circuit
- Test on a Simulation Environment.

Real Radiation Tests on the Circuit is a technique also called Hardware Fault Injection without Contact. In this technique, a circuit is exposed to real radiation. It is very precise in assessing the weak points of the circuits, but it requires a high cost and can only be afforded by Space Agencies. An approach is to use a particle accelerator to simulate cosmic rays on the circuit. A cheaper alternative that has gained widespread acceptance in the radiation-effect community has been the use of laser pulses to simulate the effects of energetic particles.

On the other hand, the Test on a Simulation Environment makes the research process easier by reducing costs. This methodology can be carried out using a Hardware or a Software environment. In this way, some previous works have been reported:

- The ESA System Division, together with the University of Seville, has developed the **FT-UNSHADES** project, which consists of a board with two Xilinx FPGA's [5].

1-4244-0755-9/07/$20.00 ©2007 IEEE

- The Politecnico di Torino has developed a new partial reconfiguration-based fault injection system to evaluate the SEU effects in SRAM–based FPGAs [6] [7].
- The LAAS-CNRS research centre, in Toulouse, has implemented the **Messaline4** [8] system, which performs failure injection through pins.
- The Chalmers University of Technology, in Sweden, has created the **FIST5** [9]. This system can apply failure injection through hardware.
- The Technical University of Wien is working towards a distributed fault-tolerant architecture, **MARS6** [10] (*Maintainable Real-Time System*).

The second approach uses simulation through software environments. In this way, 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. There are also some previous works in this area:

- The South Florida University, together with Honeywell Space System Inc., has developed a spatial redundancy method to protect circuits against SEU's [11].
- The University of Texas at Austin has created **Ferrari7** *(Fault and Error Automatic Real-Time Injection*), which is a software based simulation system that injects errors in circuits through the generation of traps [12].
- The European Space Agency (ESA) has developed a first release of a tool called SST. It works on Modelsim 5.8 and allows the insertion of SEE in a flexible way [13].

The technique proposed in the present paper is framed within the software-based simulation mechanisms described before.

III. DESCRIPTION OF THE SIMULATION PLATFORM

The purpose of this platform [14] is to help the designers to predict and explore potential weak points on ICs sensitive to hazards. It can also be used to assess the effectiveness of a given protection technique. The platform is composed of the SST (that is used for soft error insertion) and Matlab (that is used to generate the reference input and output data). A commercial circuit environment (Modelsim) is used to run the simulations. For a given circuit under test, we would generate a number of test cases in terms of the corresponding input and output data using Matlab. We would also generate a number of test configurations in terms of the soft errors inserted using the SST. The test cases would be designed to fully test the circuit functionality and performance while the test configurations would ideally reflect the soft error environment that is expected for the device operating conditions. Then, combinations of both can be easily tested by just selecting the appropriate input and output data files and test configuration file. In fact, with a simple script, the testing of all relevant combinations can be easily automated. Next, we continue explaining in depth the three independent modules which make up the platform (Fig. 2):

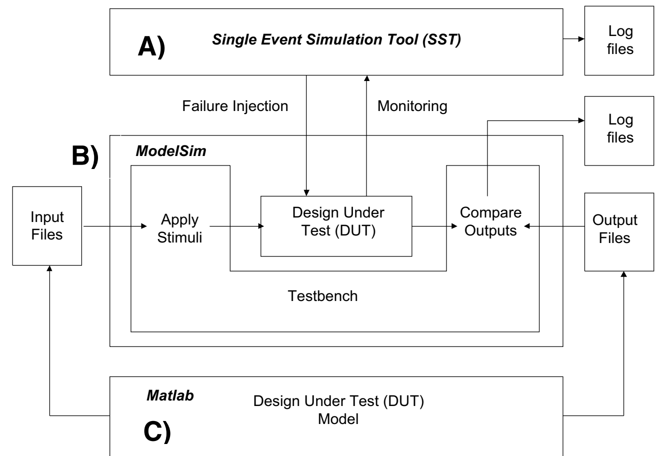


Fig. 2: Proposed Error Simulation Platform

A) SEUs Simulation Tool (SST). The initial version of the SST was developed in 2004 at the European Space Research and Technology Centre (ESTEC) and consists of a set of Perl and Tcl scripts used to prepare the environment to be able to produce upsets (bit flips). Perl is the main programming language of the tool, and Tcl/Tk is used to interact with the simulator (Modelsim). The first release worked with Modelsim 5.8 and allowed the user to upset the circuit with independence of the particular design (and in a non-intrusive way), on any register or internal signal of the DUT, while a simulation is running.
Recently our research group has carried out some tasks over the SST that improves and extends the original software:

- First enhancement: the SST has been redesigned to support the last Mentor Graphics's simulator version (Modelsim 6.1).
- Second enhancement: some functional upgrades have been added in order to make the tool a single access point for the designer. In this way, the tool scripting module may automate a great variety of tests where it can combine not only the failure injection, but also other required features to complete the simulation process like checkpoints, restores and comparisons (Fig. 3).

We keep on describing the multiples alternatives supported by the SST tool, explaining the differences between the "Injection" and "Simulation" commands.
The "Injection Commands" help to setup the way to insert the fault model:

- *Manual Option*. The user has to edit manually the instances and the wire files previously created gathering the information of the design under test, in order to select the desired wires and the upset times. This option excludes the random SEU generation.
- *Instances Option.* This option is used to specify in what instances, from the ones that can be found in the instances

file, we want to produce the upsets. We can select the instances filtering their names using Perl patterns.

- *Number Option*. This option is used to set which wires will be upset randomly. We can also select the wires filtering their names using Perl.
- *Time option*. This allows us to configure a time window where the software can insert SEUs randomly.

The Simulation Commands described next, add more flexibility to the campaign simulation offered by the SST. Both functions can be combined among them and with the injection commands:

- *Checkpoint Option*. It saves the state of a simulation with or without SEUs in order to restore the state or to compare with another simulation at a later time.
- *Restore Option*. It restores the state of a simulation that was saved with the checkpoint option. Checkpoint/restore allows a cold restore, followed by simulation activity where SEUs can be injected.
- *Compare Option*. It can perform a comparison between a gold and a test simulation or on the other hand, two test case simulations.

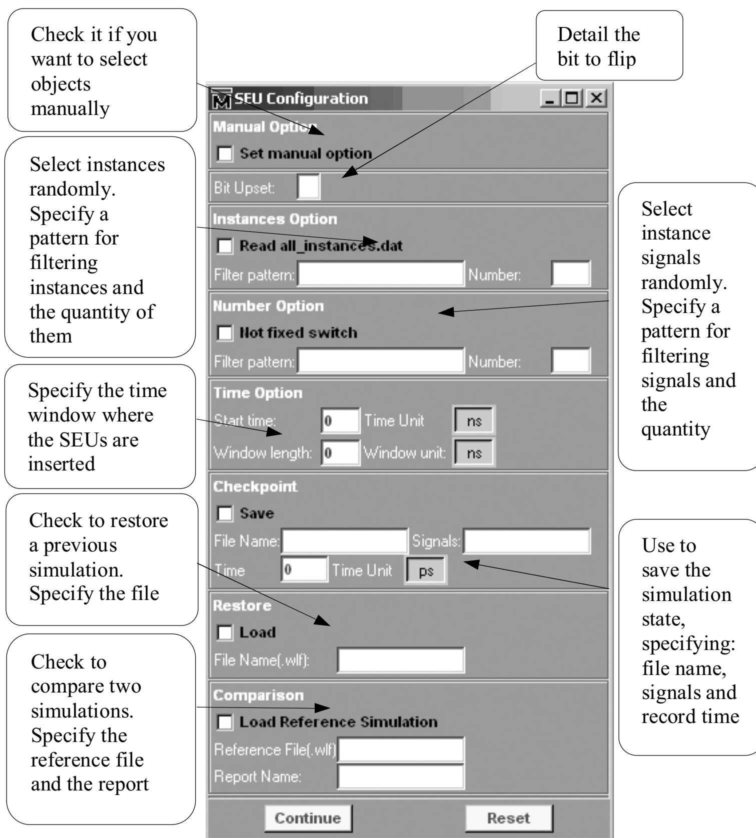


Fig. 3: Interface Functionalities

B) Modelsim 6.1 SE Design Environment (by Mentor Graphics). This standard environment, widely used in the industry, allows circuit specifications in several Hardware Description Languages, as VHDL. This module is in charge of holding the circuit to test, and performing a simulation at the design stage. The description of the environment is divided in two parts:

- The circuit itself.
- The test bench. This test bench will produce the different test scenarios for the circuit (based on the input values provided by Matlab), will capture the circuit outputs, and will compare them with the expected results (also provided by Matlab). In case both are different, that will indicate an error, which will be logged in the system for further study. This environment is generic (independent of the circuit behaviour) for circuits devoted to signal processing (or at least a significant part of them). It is also flexible in the way that it is straightforward to generate different input signals to test the circuit operating in several environments. For other kinds of circuits (e.g., controllers), another application rather than Matlab would be designated to hold the golden data.

C) Matlab. Through this module, the theoretically correct behaviour of the system will be foreseen, which will be compared to the actual output produced by the system in Modelsim. It has the advantage that the Matlab code does not need to reflect the actual circuit implementation; it only needs to be functionally equivalent. This facilitates the use of a single Matlab model to explore different implementation alternatives. The difference between both behaviours will indicate the presence of a SEU, what will trigger the mechanism to detect the source of such an error. As mentioned before, the use of Matlab is justified for signal processing circuits as it is normally utilized for the design of this kind of applications. Other kinds of circuits will be treated with a more appropriate application.
As a summary, this is the work flow of the platform (Fig. 4):

1. Load the circuit (VHDL) with a previously verified behaviour. In the other case, restore an earlier simulation from the SST.
2. Apply the stimuli to the circuit and insert a SEE somewhere in it.
3. Gather the circuit behaviour after the SEE.
4. Produce the real behaviour (golden) through Matlab. Also run a golden simulation from the SST or restore a previous one.
5. Compare the output of 3 and 4 in two levels: behaviour and simulation. If different, the discrepancies are reported by the SST.
6. Repeat from 2 until all the experiments are done.
7. If any error has been logged, decide whether
   a. to apply TMR.
   b. to use a more tolerant implementation, among the existing ones.
   c. to design an “ad hoc” protection technique. This usually implies less cost in area than the other alternatives.

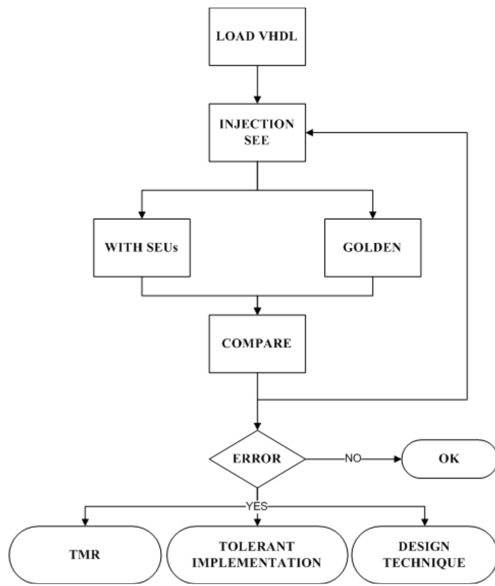


Fig. 4: Work Flow

IV. CASE STUDY

As an example of use, a simple case study is presented in which we assess the behaviour of two different structures of moving average filters [15] when soft errors are inserted. A moving average filter implements the following equation:

$$y[n] = \frac{1}{N}\sum_{i=0}^{N-1} x[n-i]$$

Where normally N is a power of 2 so that the division is just a shift operation. The structures that we are going to analyze, traditional and recursive, are both shown in Fig. 5.

The top Fig. 5 represents a traditional scheme. It can be seen that any SEU in this filter will have a transient (temporary) effect. The reason of this is because no matter which flip-flop

Error! No topic specified.

Fig. 5: Traditional structure (top) and Recursive structure (bottom)

is hit by the bit flip, its effect will only last until the content of such flip-flop is shifted out of the circuit. Note that since the delay line is shifted one position to the right for each clock cycle, the effect of a SEU will be present at most a number of cycles equal to the number of filter taps (worst case).

On the other hand, the bottom Fig. 5 represents a recursive structure. This is much more efficient in terms of area (less adders), but SEUs can have a permanent effect. Note that although the delay line is also shifted right, the output of the filter is accumulated in another register making any SEU effect permanent until the whole structure is reset.

The first experiment will be to excite both filters with a common input signal (Fig. 6-top). Since no SEUs are present, the expected output (behavior) should be the same in both cases. This can be seen in Fig. 6-bottom: both behaviors coincide.

Subsequently, using the platform we propose the following test model:

- Generate the test process from the SST, inserting a SEU which is defined by Tap-Number, Register Bit (MSB/LSB for this example) and Simulation-Time for both filters
- Analyze the results offered by the SST: by comparing the filter outputs for both structures when they suffer the same soft error.

These results are depicted for the traditional structure (FIR) in Fig. 7. In this case, we show some screenshots where the platform operation for the campaign and post-analysis is presented in detail.

Fig. 7-1 shows the SST main screen where the SEU configuration is setup to carry out the test. Each option was mentioned previously in Fig. 3. Fig. 7-2 relates to the same SST "Simulation Commands" screen. In this example it performs a comparison between the current test (Fig. 7-1 simulation with a SEU in the most significant bit, MSB, register 1) and a previously saved simulation where the SEU was inserted in the least significant bit (LSB) of register 1, both over FIR structures. Fig. 7-3 shows the differences between the outputs for both simulations from the Modelsim wave window in a graphical way.

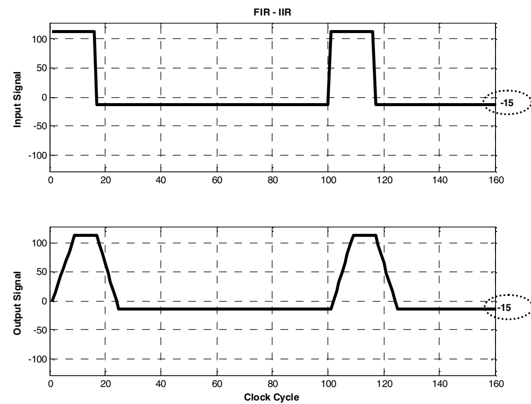
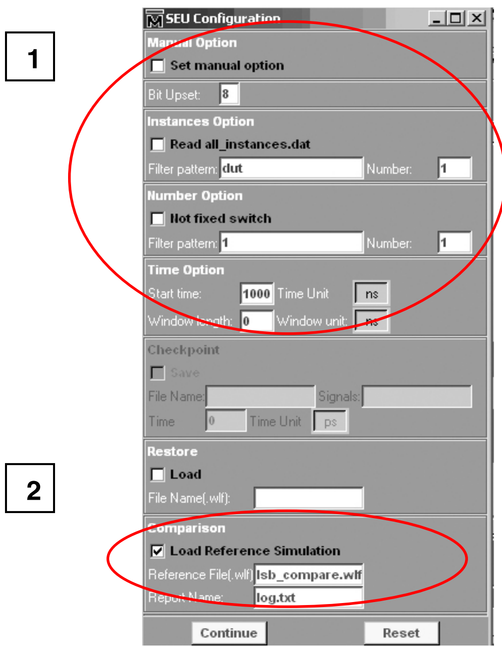


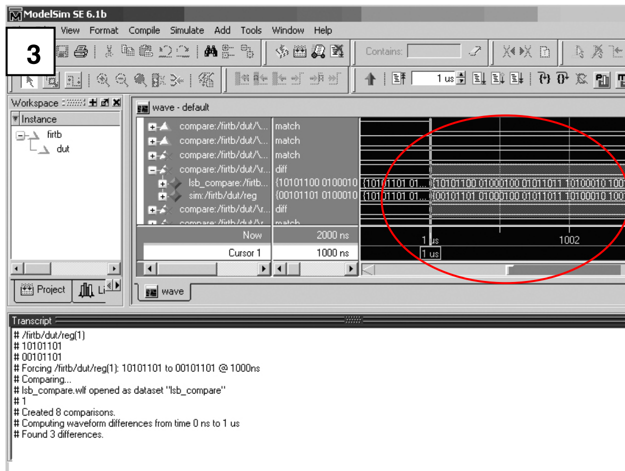
Fig. 6: Top-Input, Bottom-Output

The first result is that the effect of the SEU in the LSB is negligible: the behaviour does not differ from the correct one, since the magnitude of the error is very small (Fig. 8-1). Finally, the SST produces a report file (Fig. 7-4) where the differences highlighted in Fig. 7-3 can be analyzed.

However, for the case of the MSB (Fig. 8-2), a slight disturbance can be seen when the SEU is applied (around cycle 10), and then the behaviour is recovered automatically. In this way, the output at the last clock cycle reaches the correct level of -15.



SST (SEU Configuration Screenshot)



Modelsim 6.1 Simulation Screenshot

4

REG1 = 10101101	LSB-SEU = {1010110**0** … 01011100}
	MSB-SEU= {**0**0101101 … 01011100}

Fig. 7: Setup & Run Injection

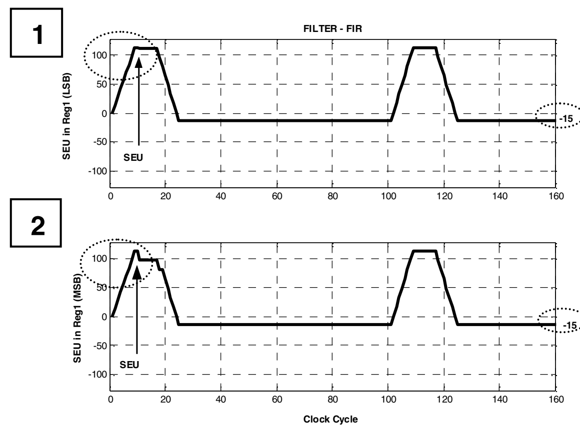


Fig. 8: Matlab plots: 1) SEU in LSB. 2) SEU in MSB

Now the same test is repeated for the recursive structure (IIR). Both results are shown in Fig. 9.

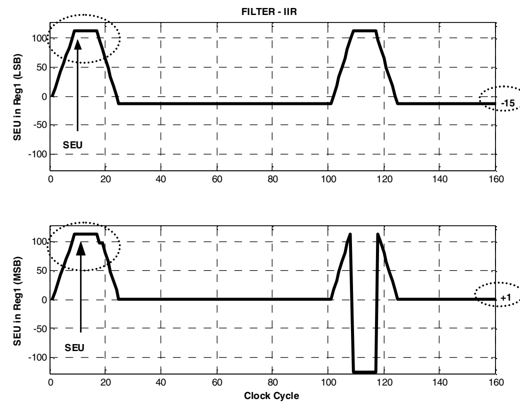


Fig. 9: Top-LSB – Bottom-MSB

Again, the first conclusion is that the effect of the SEU in the LSB is negligible (Fig. 9-top). However, when the SEU affects the MSB, a clear effect is detected. The filter gets a wrong behaviour and does not recover from it. This can be seen because the output at the last clock cycle (Fig. 9-Bottom) has a value of +1, when the correct output (Fig. 6-Bottom) has a value of -15. In others words, the SEU shifts up the whole graph in 16 units. Another side effect happens during cycles 110 to 117. The accumulator is overflowed by the effect of the SEU and that is why the filter shows such an erratic behaviour.

The last experiment studies the effect of a SEU into the recursive filter accumulator. The accumulator is also sensitive to this kind of problems, and an error there has worse consequences, since the output of the filter is directly the upper bits of the accumulator.

The results can be seen in Fig. 10. Again, a SEU on the LSB has little impact (Fig. 10-Top). However, a SEU on the MSB produces a clearly distorted result (Fig. 10-Bottom), as expected.

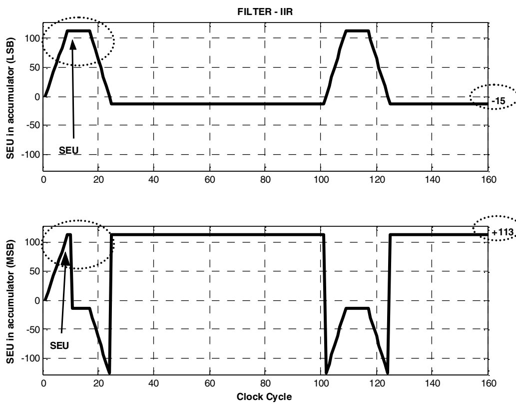


Fig. 10: Top-LSB – Bottom-MSB

As part of the case study evaluation, the performance of the platform in terms of simulation time has been measured. The results are presented in Table I and correspond to one random SEU by iteration (simulation) of 10K clock cycles each, running on a PC with an Intel(R) Pentium(R) M processor at 2.00 GHz.

TABLE I
PLATFORM PERFORMANCE (IN MILISECONDS)

Number of Iterations	10	50	100
FIR	6713,6	34359,3	82732,6
FIR with TMR	7315,2	40499,0	116739,0

These execution times show the convenience of the platform to conduct simulation experiments on medium-size circuits, since the time overhead introduced is acceptable, when compared with the flexibility offered by this technique.

V. CONCLUSIONS AND FUTURE WORK

The benefits that simulation environments can offer have been analyzed using a new platform. Through the integration into commercial design flows and due to the capacity of scripting, this proposal increases the ability to test fault tolerant technologies without introducing a large overhead in resources and time. Thanks to the software-based simulation, this platform is affordable to most research groups that do not have access to costly hardware and equipment.

We foresee that the methodology can improve the design flow in the following areas: earlier detection of errors, reduction of time-to-market and also the ability to quickly evaluate new protection techniques.

As an example of use of the platform, the fault-tolerance properties of two filters have been tested, being able to determine that the traditional structure (although more costly in area), is more resilient to errors than the recursive one.

The future work will be oriented in two different lines:

- The SST will be coupled with Matlab in such a way that the first provides the dimensions for errors in terms of time instants and nodes and the second implements the error model and produces the pairs of time and node for error insertion that are then fed back into the SST as well as helping to maintain the independence of the circuit implementation.

- Use the platform to study the behaviour of other circuits in the presence of radiation and also to design new protection techniques for those circuits (e.g. the 8051 controller).

VI. REFERENCES

[1] D. C. Mayer and R. C. Lacoe. “Designing Integrated Circuits to Withstand Space”, The Aerospace Corporation Magazine of Advances in Aerospace Technology, vol 4, no 2, Summer 2003.

[2] P.E. Dodd and LL. Massengill, “Basic Mechanisms and Modeling of Single-Event Upset in Digital Microelectronics”, IEEE Transactions on Nuclear Science Vol 50, No 3, June 2003.

[3] J. E. Mazur, “An Overview of the Space Radiation Environment”, The Aerospace Corporation Magazine of Advances in Aerospace Technology’, vol. 4, no 2, Summer 2003.

[4] R.D. Schrimpf and D.M. Fleetwood, “Radiation effects and soft errors in integrated circuits and electronic devices”, World Scientific Publishing, 2004. ISBN: 981-238-940-7.

[5] M. Aguirre et al., “An FPGA based hardware emulator for the insertion and analysis of Single Event Upsets in VLSI Designs”, Procs. of RADECS’04.

[6] L. Sterpone, M. Violante and S. Rezgui, “An Analysis based on Fault Injection of Hardening Techniques for SRAM-based FPGAs”, IEEE Transactions on Nuclear Science, Vol. 53, Issue 4, August 2006.

[7] M. Violante, L. Sterpone, M. Ceschia, D. Bortolato, P. Bernardi, M. Sonza Reorda and A. Paccagnella, “Simulation-Based Analysis of SEU Effects in SRAM-based FPGAs”, IEEE Transactions on Nuclear Science, Vol. 51, no. 6, Dec. 2004.

[8] J. Arlat et al., “Fault Injection for Dependability Validation: A Methodology and Some Applications”, IEEE Transactions on Software Engineering, vol 16 no 2, February 1990.

[9] O. Gunnetlo, J. Karlsson, and J. Tonn, “Evaluation of Error Detection Schemes Using Fault Injection by Heavy-ion Radiation”, Proc. 19th Ann. Int’l Symp. Fault-Tolerant Computing, IEEE CS Press, Los Alamitos, Calif., 1989.

[10] J. Karlsson, J. Arlat, and G. Leber, “Application of Three Physical Fault Injection Techniques to the Experimental Assessment of the MARS Architecture”, Proc. Fifth Ann. IEEE Int’l Working Conf. Dependable Computing for Critical Applications, IEEE CS Press, Los Alamitos, Calif., 1995.

[11] N. Rollins, M. Wirthlin, P. Graham and M. Caffrey, ”Evaluating TMR Techniques in the Presence of Single Event Upsets”, 2003 MAPLD International Conference, September 2003.

[12] G. A. Kanawati, N. A. Kanawati and J. A. Abraham, “FERRARI: A Tool for the Validation of System Dependability Properties”, Proc. 22nd Ann. Int’l Symp. Fault-Tolerant Computing, IEEE CS Press, Los Alamitos, Calif., 1992.

[13] D. Gonzalez-Gutierrez, “Single Even Upset Simulation Tool Functional Description”, ESA Report TEC-EDM/DCC-SST2, July 2004.

[14] Computer Architecture and Technology Group, Universidad Antonio de Nebrija. Available: http://www.nebrija.es/~jmaestro/esa

[15] A. V. Oppenheim and R. W. Schafer, “Discrete Time Signal Processing”, Prentice Hall, 1999.