Analysis of SEU effects in a pipelined processor

M. Rebaudengo, M. Sonza Reorda, M. Violante
Politecnico di Torino
Dipartimento di Automatica e Informatica
Torino, Italy
www.cad.polito.it

Abstract*

Modern processors embed features such as pipelined execution units and cache memories that can hardly be controlled by programmers through the processor instruction set. As a result, software-based fault injection approaches are no longer suitable for assessing the effects of SEUs in modern processors, since they are not able to evaluate the effects of SEUs affecting pipelines and caches. In this paper we report an analysis of a commercial processor core where the effects of SEUs located in the processor pipeline and cache memories are studied. Moreover, the obtained results are compared with those software-based approaches provide. Experimental results show that software-based approaches may lead to errors during the failure rate estimation of up to 400%.

1. Introduction

The need for developing safety or mission critical applications with high demands in terms of computational power under low-cost constraints pushed designers to explore the possibilities offered by commercial high-performance processors. Commercial processors are bundled with features allowing them to execute a very high number of instructions per second; moreover, mostly thanks to the high number of sold units, their cost is orders of magnitude less than that of their hardened versions. They are thus particularly attractive since they combine low cost with high performance.

In order to fruitfully exploit them in critical applications, their sensitivity to soft errors should be deeply investigated. In particular, detailed characterization of processor behavior in presence of Soft Event Upsets (SEUs) is mandatory. In order to cope with the complexity of the task, which mandates the ability of altering and monitoring the operations of a processor during the execution of a workload, hardware prototypes are usually exploited. Radiation testing is then normally adopted to analyze SEU effects on microprocessors in terms of static cross-section. Static cross-section corresponds to the sensitivity to SEUs of all the processor memory elements (registers and internal memories) and is independent on the executed program. In practice, static cross-section is often obtained by measuring the number of corrupted bits in the processor storage elements while the circuit is exposed to suitable radiation beams. Static cross-section is then combined with the figures characterizing the final environment to estimate the error rate of the final application. The obtained figure is a worst-case estimation of SEU effects because it does not take into account the impact of the executed application on the processor cross-section: for example an application may use only a limited portion of the processor register file, and thus SEU effects on the un-used registers should be ignored during cross-section computation.

Recently, an alternative approach [1] has been proposed to overcome this limitation: the method combines software fault injection results with static cross-section figures derived from radiation experiments. In particular, static cross-section is multiplied by the ratio of injected faults producing a failure (hereinafter called *failure rate*), obtaining the application error rate.

In the outlined analysis process, failure rate computation through fault injection is crucial for obtaining accurate estimations of the application error rates. As far as simple processors are concerned, software-based fault injection [2] is commonly adopted for this task. It indeed allows injecting faults while running the application at the processor full-speed; a high number of faults can thus be analyzed in a reasonable amount of time.

When modern high-performance processors are concerned, software-based fault injection is no longer able to provide accurate estimation of the processor failure rate. These processors employ architectural solutions such as pipelined execution units, out-of-order instruction issue units and cache memories that significantly increase the number of memory elements the processor embed, and that are hidden to programmers, since no instruction is available to access them. As a result, software-based fault

* This work has been partially supported by Center of Excellence on Multimedia Radiocommunications (CERCOM) of Politecnico di Torino, by Agenzia Spaziale Italiana and by the ISIDE project.

injection cannot be exploited to evaluate SEUs effects in these un-reachable (hidden) memory elements.

On the other side, simulation-based fault injection [3][4] can be exploited to gather more detailed information about fault effects, but its adoption is strongly limited by the enormous amount of CPU time it requires.

The purpose of this paper is to experimentally evaluate the effects of SEUs affecting the processor-hidden parts. For this purpose we exploited the fault injection environment described in [5], which provides a very efficient way of assessing the effects of SEUs in all the processor memory elements including those that are not accessible via software-based fault injection. On the other side, the approach provides high efficiency in terms of required fault injection time; it thus combines the advantages of simulation-based fault injection with the speed of the software-based one.

In our analysis we considered a processor implementing the Sparc v8 architecture running some benchmark programs: for each of them we compared the failure rate measured when performing two experiments:

1. We injected SEUs in the processor register file as usually did when software-based fault injection is exploited.
2. We extended SEU fault injection to processor pipeline and cache memory elements.

Faults have not been injected in the data/code memory since we are interested in studying the effects of SEUs on the processor internal memory elements, only. The aforementioned experiments allow analyzing the effects of SEUs in a processor core, as well as to gather results for comparing software-based approaches with new, more accurate, approaches.

The preliminary results reported in the paper show that, when pipelined processors are considered, failure rate computation couldn't neglect the effects of SEUs in the processor hidden part. Moreover, they show that the proposed fault injection approach is more accurate than traditional software-based approaches for computing the actual failure rate of applications running on modern microprocessors.

The paper is organized as follows. Section 2 shortly summarizes the fault injection environment adopted in this paper, and section 3 discuesses the considered fault model. Section 4 describes the analyzed processor, while section 5 reports the obtained results. Finally, section 6 draws some conclusions.

2. Fault injection environment

The exploited FPGA-based fault injection environment is described in figure 1. A software tool instruments the model of the analyzed processor according to the instrumentation mechanisms described in [5]. The obtained model is then synthesized and mapped on a FPGA device.

Two hardware platforms are used: a host computer and a FPGA board. The former acts as a master and is in charge of managing Fault Injection campaigns. The latter acts as a slave and is used to emulate the system under analysis. In particular, the FPGA-based fault injection environment exploits a FPGA board where two modules are implemented: the emulated system and the fault injection interface, which allow a host computer to control the behavior of the emulated system.

A major advantage of the adopted fault injection environment is that it operates on RT-level processor descriptions, which are more widely available than gate-level ones.

The fault injection environment is composed of the following modules:

- *Fault List Manager*: this module is in charge of generating the list of faults to be injected according to the system, the input data and the possible indications of the designer (e.g., information about the most critical portions in the system).
- *Fault Injection Manager*: this module is the core of the fault injection environment, and is in charge of orchestrating the selection of a new fault, its injection in the system, and the observation of the resulting behavior.
- *Result Analyzer*: the task of this module is to analyze the output data produced by the system during each fault injection experiment, categorize faults according to their effects, and produce statistical information.

The FPGA board is driven by a host computer where the other modules and the user interface run. The fault injection manager is implemented as a hardware/software system where the software partition runs on the host, and the hardware partition is located on the FPGA board along with the emulated system.

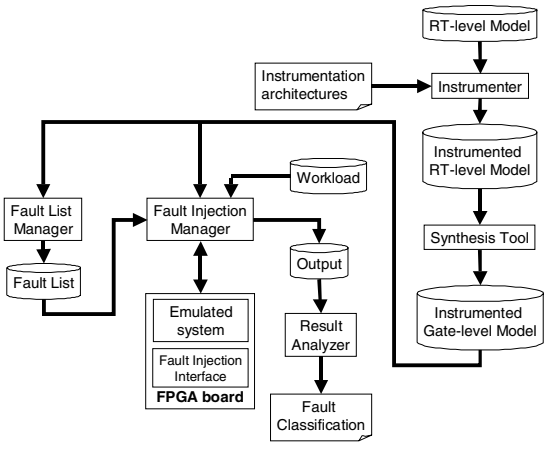

Figure 1: The FPGA-based fault injection flow

To efficiently determine the behavior of the circuit when a fault appears, the FPGA board emulates an in-

strumented version of the system, which allows both the injection of each fault, and the observation of the corresponding faulty behavior.

3. The fault model

The fault tolerance community is increasingly concerned by the occurrence of soft errors resulting from the perturbation of storage cells caused by ionization. This type of soft errors is known as *Single Event Upset* (SEU). A characteristic of SEUs is that they are random events and thus they may occur at unpredictable times. For example, they may corrupt the content of a processor register during the execution of an instruction.

In this paper we focused on the fault model called *upset* or *transient bit-flip*, which results in the modification of the content of a storage cell during program execution. Possible fault locations are thus memory cells, flip-flops, bits of user registers, control registers and even registers not accessible through the processor instruction set, and embedded memories such as register files and caches.

Despite its relative simplicity, the bit-flip is widely used in the fault tolerance community to model real faults since it closely matches the real faulty behavior [6].

4. Analyzed processor

We considered a core implementing the SPARC v8 architecture [7]. The core description corresponds to about 100,000 lines of synthesizable RT-level VHDL code. The model includes a 2Kbyte instruction cache and a 2Kbyte data one, an integer multiplication and division units, and a 5-stage pipeline. When synthesized, this description produces a netlist of about 35,000 gates. The core has been instrumented according to the flow described in the previous section and then synthesized on a Xilinx Virtex 1000E device. The obtained design amounts to 4,762 out of 12,288 logics blocks, uses 14 out of 96 block RAMs and runs at 30 MHz.

The architecture of the core is reported in figure 2, where the processor memory elements are highlighted.

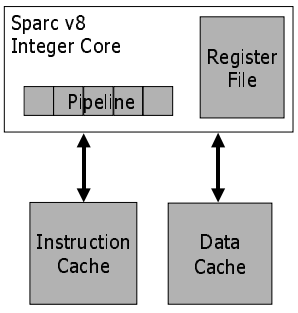

Figure 2: Architecture of the considered processor.

By analyzing the reports produced by the Synopsys FPGA Compiler II tool, we gathered the number of flip-flops in the circuit, which are summarized in table 1.

Processor module	Number of flip-flops
Pipelined integer unit	742
Register file	8,192
Cache-memory	4,096

Table 1: Processor memory elements

As the reader can observe from table 1, the processor has a significant amount of flip-flops that cannot be controlled by the programmer. Indeed, the processor instruction set does not provide instructions for accessing the memory elements inside the pipeline, neither instruction for controlling the memory implementing the cache, which account for about 40% of the processor internal memory.

5. Experimental results

In our experimental analysis we considered two benchmark programs:

- *MTX*: a matrix multiplication program, where two 4x4 integer matrices are multiplied.
- *FFT*: the algorithm adopted in digital signal processing applied to a 4-point network.

The programs, whose characteristics are reported in table 2, are coded in C and have been compiled resorting to the GNU C/C++ `gcc` compiler, which is able to generate code for the LEON processor [7].

Program	Number of C lines	Data segment size [byte]	Code segment size [byte]	Execution time [# clock]
MTX	35	16	2,832	5,033
FFT	512	784	21,936	174,557

Table 2: Programs characteristics

In order to assess the effects of SEUs on the processor internal memory elements, we exploited the fault injection environment described in section 2. Five fault injection campaigns have been performed:

- *RF*: 10,000 faults are injected in the processor register file. Fault locations are randomly selected among the bits of the register file, while injection times are randomly selected among the clock cycles needed for executing the program.
- *PIPELINE*: 10,000 faults are injected in the processor pipeline, only. Fault locations are randomly selected among the bits of the registers implementing the processor pipeline, while injection times are randomly selected among the clock cycles needed for executing the program.

- *D-CACHE*: 10,000 faults are injected in the processor data chache, only. Fault locations are randomly selected.
- *I-CACHE*: 10,000 faults are injected in the processor instruction chache, only. Fault locations are randomly selected.
- *ALL*: 10,000 faults are injected in the whole processor (register file, pipeline and cache memories). In this experiment, fault locations are randomly selected.

These experiments are based on the assumption that the register file, cache memories and the processor pipeline are implemented with the same manufacturing technology. As a result, they have the same probability of being affected by charged particles hitting the circuit.

Results gathered on the benchmarks during the fault injection campaigns are reported in table 3, where the time for performing each fault injection campaign is reported, too.

Fault effects are classified according to three broad categories:

1. *Wrong answer (WA)*: the results produced by the faulty processor are different than those produced by the fault-free processor.
2. *Effect-less (EL)*: the results produced by the faulty processor are equal to those produced by the fault-free processor.
3. *Time-out (TO)*: the faulty processor is not able to produce the expected result after a given amount of time.

Bench.	Campaign	WA	EL	TO	Time [s]
MTX	RF	180	9,790	30	515
	PIPELINE	1,000	8,760	240	526
	D-CACHE	250	9,750	0	512
	I-CACHE	540	9,370	90	515
	ALL	294	9,659	47	518
FFT	RF	220	9760	20	11,957
	PIPELINE	800	8,910	290	12,232
	D-CACHE	1,260	8,720	20	11,985
	I-CACHE	5,550	3,840	610	11,908
	ALL	1,255	8,618	127	12,050

Table 3: Fault injection results

Given the figures of table 3, it is possible to compute the error rate [1] for the system. The error rate can be computed as follows:

$$\tau_{SEU} = \sigma_{SEU} \cdot P_{WA} \qquad (1)$$

where σ_{SEU} is the SEU static cross section (in $cm^2 / device$) of the considered processor, and P_{WA} is the probability for SEUs to produce wrong answer.

We computed the latter term from table 3 as the number of wrong answers over the number of injected faults, while σ_{SEU} is measured by performing a static test: the content of the processor memory elements is continuously read during a radiation session. Static cross section thus measures the fraction of particles hitting the circuit that originates SEUs in the processor memory elements.

By exploiting Eq. (1), we obtained the figures reported in Table 4, where $\sigma_{SEU} = 2.28 \cdot 10^{-4} cm^2 / device$.

Bench.	Campaign	Error rate [$cm^2 / device$]
MTX	RF	$4.10 \cdot 10^{-6}$
	PIPELINE	$22.80 \cdot 10^{-6}$
	D-CACHE	$5.70 \cdot 10^{-6}$
	I-CACHE	$12.30 \cdot 10^{-6}$
	ALL	$6.70 \cdot 10^{-6}$
FFT	RF	$5.02 \cdot 10^{-6}$
	PIPELINE	$18.20 \cdot 10^{-6}$
	D-CACHE	$28.70 \cdot 10^{-6}$
	I-CACHE	$127.00 \cdot 10^{-6}$
	ALL	$28.60 \cdot 10^{-6}$

Table 4: Error rates

These results show that, when all the processor memory elements are affected by charged particles originating SEUs, the processor failure rate increases over the same figure recorded when only the register file is considered. In our experiments we observed an error rate figure 63% higher when a simple application such as the MTX one is considered; while for larger applications such as the FFT one, we observed an error rate figure more then 400% higher. Moreover, the number of TO faults is more than doubled. These figures also indicate that the registers in the processor hidden parts (pipeline and cache memories) are particularly susceptible to radiation effects.

This preliminary result suggests that, when complex processors are considered, the hidden memory elements (i.e., memory elements that are not accessible through the instruction set) may significantly modify the figures coming from fault injection experiments. Fault injection techniques providing access to hidden memory elements are thus mandatory to produce meaningful information about the effects of SEUs on modern processors. In particular, the needed fault injection environments should conjugate flexibility in terms of fault location, to provide access to all the processor memory elements, with high efficiency, to inject high number of faults in a reasonable amount of time.

6. Conclusions

This paper analyzed from an experimental point of view the effects of SEUs in the pipeline and cache memories of a modern processor core. Thanks to the availability of a fault injection environment that provides full access to all the memory elements the processor embeds, we were able to analyze the effects of SEUs hitting the processor with higher accuracy than that software-based fault injection approaches may provide. In particular, the adopted environment allowed us to inject faults inside the processor pipeline as well as its register file and its cache memories, while software-based approaches provide access to the register file, only.

Preliminary results gathered on a Sparc processor description running a simple benchmark program show that by neglecting the hidden memory elements, errors as high as 400% might be obtained during the estimation of the processor failure rate. These results thus suggest that software-based fault injection is no longer suitable to address the analysis of processors embedding a high number of hidden memory elements.

7. References

[1] R. Velazco, S. Rezgui, R. Ecoffet, "Predicting error rate for microprocessor-based digital architectures through C.E.U. (Code Emulating Upsets) injection", IEEE Transactions on Nuclear Science, Vol. 47, No. 6, 2000, pp. 2405-2411

[2] R. K. Iyer and D. Tang, Experimental Analysis of Computer System Dependability, Chapter 5 of Fault-Tolerant Computer System Design, D. K. Pradhan (ed.), Prentice Hall, 1996

[3] B. Parrotta, M. Rebaudengo, M. Sonza Reorda, M. Violante, "New Techniques for Accelerating Fault Injection in VHDL descriptions", IEEE International On-Line Test Workshop, 2000, pp. 61-66

[4] J. Garcia, J. C. Baraza, D. Gil, P. J. Gil, "A Study of the Experimental Validation of Fault-Tolerant Systems using different VHDL-Based Fault Injection Techniques", IEEE International On-Line Test Workshop, 2000, pp. 140

[5] P. L. Civera, L. Macchiarulo, M. Rebaudengo, M. Sonza Reorda, M. Violante, "FPGA-based Fault Injection for Microprocessor Systems", IEEE Asian Test Symposium, 2001, pp. 304-309

[6] M. Nikoladis, "Time Redundancy Based Soft-Error Tolerance to Rescue Nanometer Technologies", IEEE 17th VLSI Test Symposium, April 1999, pp. 86-94

[7] http://www.gaisler.com