An Accurate Analysis of the Effects of Soft Errors in the Instruction and Data Caches of a Pipelined Microprocessor

M. Rebaudengo, M. Sonza Reorda, M.Violante

Politecnico di Torino
Dipartimento di Automatica e Informatica
Torino, Italy
http://www.cad.polito.it/

Abstract*

Instruction and data caches are well known architectural solutions that allow significantly improving the performance of high-end processors. Due to their sensitivity to soft errors they are often disabled in safety critical applications, thus sacrificing performance for improved dependability. In this paper we report an accurate analysis of the effects of soft errors in the instruction and data caches of a soft core implementing the SPARC architecture. Thanks to an efficient simulation-based fault injection environment we developed, we are able to present in this paper an extensive analysis of the effects of soft errors on a processor running several applications under different memory configurations. The procedure we followed allows the precise computation of the processor failure rate when the cache is enabled even without resorting to expensive radiation experiments.

1 Introduction

New application areas, such as the automotive one that is pursuing the drive-by-wire philosophy, are fostering the adoption of high-performance processors in safety-critical applications. In this areas two conflicting constraints should met. On the one hand, there is the need for adopting modern processor cores able to support complex operations in a real-time context, such as the control of active shock absorber systems or steer-by-wire. Moreover, the need for keeping the product cost as low as possible is making commercial-off-the-shelf processors very active. On the other hand, these application areas are characterized by tight dependability requirements: since the processor-based systems are employed in applications that can harm the human life, it is mandatory for them to guarantee high dependability levels.

Commercial-off-the-shelf processors, and in particular the high-performance ones, can be exploited within safety-critical applications only when their behavior in presence of faults is known. In this context, the fault tolerance community is increasingly concerned by soft errors resulting from the perturbation of storage cells (caused for example by ionization [1]) known as Single Event Upset (SEU). Unlike memory modules that can be easily hardened by resorting to solution ranging from the simple parity bit to more complex codes, the problem of avoiding soft errors in memory modules inside high-end processors is much more complex. This kind of 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 processors embed. These memory modules are usually not hardened against SEUs and thus the dependability level the processor provides can be very low. This problem is usually addressed by simply disabling the processor cache memory. The processor area that is susceptible to SEUs is therefore drastically reduced while the processor dependability is greatly increased. Although very simple and effective from the dependability point of view, this solution implies a significant performance loss that may be not acceptable when complex computation are required, in particular when real-time constraints should be met. As a result, no simple assumptions (such as to disable cache memories) can be done and the dependability of the processor-based system should be carefully analyzed.

The effects of soft errors in modern processors, and in particular in cache memories, were already studied in several works, such as [2]-[6]. Several analysis approaches were adopted for assessing the effects of soft errors:

* This work has been partially supported by the Center of Excellence on Multimedia Radiocommunications (CERCOM) of Politecnico di Torino.

radiation testing, fault injection and static computation based on error propagation models.

The main contribution of this paper lies in the analysis procedure we propose, which offers an effective solution for analyzing processor cores without the high costs and accuracy loss other approaches may imply. The proposed approach is general, i.e., it can be adopted for studying any processor provided that a model is available, and it can be adopted for accurate analysis, both in terms of fault location and fault injection time. On the contrary, the approaches based on radiation testing lacks the ability to carefully select the fault location and the fault injection time; as a result, they do not allow to quantify the contribution to the processor dependability of each memory component (i.e., cache lines, cache tags, pipeline, register file). Moreover, the approaches proposed so far exploiting simulation-based fault injection [7] or analytical models lacks accuracy, since they resort to purely behavior processor models, thus neglecting the implementation details that may significantly alter the obtained results.

As an example of application of our approach, the paper reports an accurate analysis of a real processor core. We were able to measure the effects of soft errors in both cache lines and cache tags, and to quantify their contribution to the processor dependability with respect to the other memory elements the processor embeds (i.e., register file and pipeline). The fault injection campaigns we performed showed that soft errors in the data cache lines and tags are particularly critical, while the effects on the instruction cache lines are strictly correlated to the considered application. An interesting result is the unexpected good robustness of instruction cache tags.

The obtained figures can be exploited to guide designers to cleverly select the processor components that should be hardened, and to identify the hardening solution that best fits the fault tolerance, area and time requirements. As an example of this kind of analysis, we evaluated two processor configurations: one exploiting all the features the processor provides and one where the cache subsystem is disabled. This analysis showed that although cache memory allows improving the processor performance by a factor of about 3, it may reduces the processor dependability level by a factor up to 24 with respect to the same application ran by a cache-less processor.

The paper is organized as follows. Section 2 reports background information about the fault model we adopted, the processor core we considered and the fault injection environment we exploited. Section 3 reports the analysis methodology we exploited for computing the processor dependability; while section 4 reports the results of the experimental analysis we performed. Finally, section 5 reports some conclusions.

2 Background

This section provides the reader with background information about the fault model we considered during our analysis, the processor core we analyzed, and the fault injection environment we adopted.

2.1 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 [1]. 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 internal memory cells, flip-flops, bits of user and control registers and even registers usually not accessible through the processor instruction set, and embedded memories such as register files and caches.

In the paper we assume that the memory modules located *outside* the processor and storing the application code/data are hardened against SEUs, and thus we concentrate only on the memory elements located *inside* the processor: pipeline registers, register file and cache memory (cache lines and cache tags).

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 [8].

2.2 Analyzed processor

We considered the Leon core implementing the SPARC v8 architecture [10]. The core description corresponds to about 100,000 lines of synthesizable RT-level VHDL code.

The model includes 2 Kbytes of memory for implementing the instruction cache (I-Cache) and 2 Kbytes for the data one (D-Cache), an integer multiplication and division units, and a 5-stage pipeline.

The Leon instruction/data cache is a direct-mapped one, and it is divided into lines with 8 bytes of data. Each line has a cache tag (I-Tag, D-Tag) associated with it consisting of a tag field and one valid bit for each 4-byte sub-block. The data cache implements a write-through policy with no-allocate on write-miss.

The processor pipelined integer unit is composed of the following stages:

- *Fetch*: it loads a new instruction either from the instruction cache or the main memory, depending on the cache subsystem configuration.
- *Decode*: it decodes the instruction and reads the required operands.
- *Execute*: it executes arithmetical, logical and shift operations. It also takes care of address computation.
- *Memory*: it accesses the data cache.
- *Write*: it writes to the register file the results of any arithmetical, logical, shift or cache read instruction.

When synthesized, this description produces a netlist of about 35,000 gates. The core has been instrumented according to the fault injection environment described in section 2.3 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.

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; in table 1 we report also the percent contribution of each module to the total number of flip-flops (column Ratio).

Processor module	Number of flip-flops	Ratio [%]
Pipelined integer unit	742	1.67
Register file	4,352	9.83
D-Cache memory	19,584	44.25
I-Cache memory	19,584	44.25

Table 1: Processor memory elements

As the reader can observe from table 1, the processor embeds a significant amount of flip-flops that can be affected by SEUs.

2.3 Fault injection environment

In this paper we exploited the simulation-based fault injection [7] technique for assessing the effects of SEUs inside a processor memory elements. In order to speed-up the execution of the fault injection experiments, we exploited an extended version of the approach described in [9], where simulation-based fault injection efficiency is improved by means of emulation: the processor model is first instrumented for supporting fault injection and then implemented on a FPGA device.

The adoption of a simulation-based approach in spite of the software-based one that is usually exploited for the analysis of processors is motivated by two reasons.

The processor we considered adopts a pipelined execution unit embedding several registers. Since they are likely to be affected by SEUs, they have to be analyzed during fault injection. Software-based fault injection which resorts to the processor instruction set for fault inoculation is not suitable for our purposes since it does not provide any mean for directly reading and writing the content of the pipeline registers.

Moreover, although the Leon instruction set provides instructions for reading and writing the contents of the cache memory, and thus software-based fault injection can be exploited, the time resolution it allows is not accurate enough for performing detailed analysis of the effects of SEUs. SEUs are random both in space and time, thus they can hit the processor area anytime. Resorting to the instruction set for injecting SEUs provides a time resolution of one instruction, and thus it assumes that a SEU should hit the processor within a time corresponding to the number of clock cycle the processor needs for executing one instruction. By resorting to the approach proposed in [9] we are able to more accurately model SEU effects, since the adopted fault injection method provides a time resolution of one clock cycle.

The fault injection environment we exploited classifies fault effects according to the following categories:

1. *Wrong answer*: the results produced by the faulty processor are different than those produced by the fault-free processor.
2. *Effect-less*: the results produced by the faulty processor are equal to those produced by the fault-free processor.
3. *Latent*: the results produced by the faulty processor are equal to those produced by the fault-free processor, but at the end of the program execution, the content of the pipeline of the fault-free processor differs from that of the faulty one.
4. *Exception*: the injected fault is detected by the error detection mechanisms the processor embeds, which force the processor to generate an exception (e.g., illegal instruction exception or invalid address exception).
5. *Time-out*: the faulty processor is not able to produce the expected result after a given amount of time.
6. *Stall*: the faulty processor computes the expected results in a time greater than the faulty-free one. Examples of faults belonging to this category are those that originate an unexpected flush of the pipeline or that invalidate a valid cache line.

3 Analysis procedure

In this paper we assume that the processor dependability level is measured as its *sensitivity* to SEU effects. Radiation testing is normally adopted to analyze SEU effects on processors in terms of static cross-section [11]. 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 after 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 [11] has been proposed to overcome this limitation: the method combines fault injection results with static cross-section figures derived from radiation experiments, according to the following equation:

$$\tau_{SEU} = \sigma_{SEU} \cdot F \quad (1)$$

where σ_{SEU} is the SEU static cross section (in $cm^2/device$) of the considered processor, and F is the probability that a SEU hitting the processor produces a wrong answer (i.e., the processor is affected by the SEU in such a way that the results it provides are different from the expected ones). The static cross section can be easily computed by performing a static test, i.e., 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, and depends only on the device manufacturing technology.

Conversely, the value F, hereinafter called *failure rate*, depends on the processor architecture and the application the processor is executing, and a good estimation accuracy is essential to provide designers with meaningful dependability figures.

In order to accurately compute the failure rate of a processor which embeds heterogeneous memory components (we assume they are all manufactured with the same technology and thus share the same static cross section) we propose the following approach:

1. For each memory module i in the processor, we inject N faults and measure the module failure rate F_i as:

$$F_i = \frac{WA_i}{N} \quad (2)$$

where WA_i is the number of faults leading the program to produce wrong answer, when N faults affect the memory elements (i.e., change their value).

2. We combine the obtained failure rates to obtain the processor failure rate F_{CPU} as follows:

$$F_{CPU} = \sum_i F_i \cdot P_i \quad (3)$$

where P_i is the probability for a soft error to occur inside the memory module i, given that a SEU occurs in the processor. Please note that the probability that a SEU affects the processor is σ_{SEU}. Since we assume the same technology and the same geometry for each memory cell, we have that:

$$P_i = \frac{B_i}{B_{TOT}} \quad (4)$$

where B_i is the number of bits in the memory module i and B_{TOT} is the total number of memory bits in the processor.

The fault injection tool we adopted provides the required accuracy since it allows accessing all the memory elements the processor embeds, with a very high time resolution.

In setting-up the fault injection experiments, a crucial factor is the selection of the number of faults to inject. Given a time resolution of one clock cycle, the total number of possible faults can be computed as follows:

$$N_{TOT} = B_{TOT} \cdot CC \quad (5)$$

where CC is the number of clock cycles that are needed for completing the execution of an application. Since the number of faults that should be simulated according to equation 5 is very high, we resort to a fault sampling technique. In particular, we selected a number of faults proportional to *CC*. This assumption allows taking into account the fact that the number of particles hitting the processor is proportional to the program execution time. As a result, the total number of SEUs originated in the processor is proportional to the program execution time.

4 Experimental results

As an application of the proposed methodology we studied the effects of soft errors inside the cache memory of the Leon processor. In particular, the purpose of the experiments we performed is twofold. On the one hand, we are interested in quantifying the impact of SEUs inside the cache memory on the processor failure rate (sub-section 4.1); on the other hand, we are interested in analyzing in greater detail the effect of SEUs in the different components of the cache subsystem: cache lines and cache tag (sub-section 4.2).

4.1 Analysis of soft errors in the cache

To evaluate the effects of soft errors in the cache subsystem of the Leon processor we considered two benchmark programs, a computational intensive one and a data-transfer intensive one:

- *MTX*: it is a matrix multiplication program, where two integer matrices are multiplied. We considered two implementations of this benchmark, one working on 4x4 matrices (MTX 4x4) and another one working of 10x10 matrices (MTX 10x10);
- *HS*: it is an implementation of the heap sort algorithm, which exploits a recursive procedure. As in the previous case, we considered two implementations of this program. One working on a set of 32 integer

values (HS 32), and one working on 64 values (HS 64).

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 can generate code for the Leon processor [10].

Program	C lines [#]	Data segment size [# byte]	Code segment size [# byte]
MTX 4x4	35	192	2,832
MTX 10x10	35	1,200	2,832
HS 32	60	132	8,384
HS 64	60	256	8,384

Table 2: Program characteristics

In a first set of experiments, we injected faults in the pipeline registers, in the register file and in the instruction and data caches (both cache lines and cache tags). For each of the above modules we injected 100,000 faults in order to get statistically meaningful results. The predominance of soft errors inside the cache subsystem on the processor failure rate can be observed by analyzing the results in table 3, where we report the percent contribution of each component to the processor failure rate.

Component	MTX 4x4 [%]	MTX 10x10 [%]	HS 32 [%]	HS 64 [%]
Pipeline	4.28	1.73	1.56	1.65
Register File	6.67	4.11	3.30	1.54
D-Cache	41.77	77.17	60.06	74.42
I-Cache	47.27	17.00	34.53	22.39

Table 3: Contribution of processor components to the failure rate

The contribution of faults inside the cache is always the dominant term: such a contribution ranges indeed from 89.04% for MTX 4x4 to 96.81% for HS 64. Moreover, the contribution of D-Cache and I-Cache greatly depends on the executed program. As expected, the larger it is the amount of data the program manipulates, the larger it is the contribution to the processor failure rate of the D-Cache.

To evaluate the impact of the cache on processor dependability and performance levels, we then performed a second set of experiments, where we considered two memory configurations mimicking two possible scenarios. In the first one the cache is disabled to improve the processor dependability, resulting in longer execution times; in the second scenario the cache is enabled, thus minimizing the program execution time. Table 4 reports the program execution time for the two considered scenarios, while table 5 reports the processor failure rates computed according to equation 3.

The obtained results show that by turning off the cache we can reduce the processor failure rate by a factor ranging from about 5.46 to 24.70. The introduced time overhead is however not negligible: the program execution time is indeed increased by a factor ranging 2.56 to 4.39. Please note that the performance degradation is much lower than the benefits stemming from disabling cache memories from the dependability point of view.

Program	Exec. time Cache OFF [# clock]	Exec. time Cache ON [# clock]	Ratio
MTX 4x4	15,074	4,972	3.03
MTX 10x10	193,327	44,076	4.39
HS 32	12,300	4,813	2.56
HS 64	23,100	7,617	3.03

Table 4: Program execution time

Program	F_{CPU} Cache OFF [%]	F_{CPU} Cache ON [%]	Ratio
MTX 4x4	0.46	2.51	5.46
MTX 10x10	0.52	7.19	13.91
HS 32	0.17	2.68	15.58
HS 64	0.18	4.46	24.70

Table 5: Processor failure rates

4.2 Analysis of the cache components

The previous experiments showed that when turned on, the cache subsystem can greatly affect the processor failure rate. We thus performed a second set of experiments aiming at better analyzing the effects of SEUs inside the cache components. We injected faults in both the cache lines and the cache tags. In this case, the number of injected faults was proportional to the size of the considered memory components.

In table 6 the fault effects are reported and classified according to the categories introduced in sub-section 2.3. The rows labeled with D-/I- Cache lines refer to the effects of SEU injected in the lines of the data/instruction cache. Conversely, the rows labeled with D-/I- Tag refer to faults injected in the tag bits associated with the data and instruction cache lines.

By analyzing the data we gathered, several considerations arise:

- SEUs in D-Cache and I-Cache lines are particularly critical; in fact, most of the soft errors hitting these memory elements lead the processor to produce wrong answers.

- D-Tag memory bits are very sensible to SEUs, which may produce either wrong answers or processor stalls. The latter case is the less critical from the dependability point of view, since this kind of effect will result in just a time overhead. Nevertheless, this time overhead may be significant when real-time constraints have to be met.
- I-Tag memory bits are the most robust against SEUs, which mainly produce processor stalls, while no wrong answers were observed.

5 Conclusions

The paper proposed a method suitable for accurately analyzing the effects of soft errors in real processor cores running an application. The information provided by an accurate and efficient fault injection environment are combined to obtain the processor failure rate for a given application. Moreover, detailed analysis of the various processor components can be performed, thus obtaining accurate estimation of their contribution to the processor failure rate. By exploiting these information, designers of cores can easily identify the most critical processor components, and decide which is the better hardening solution. Finally, the approach can be used to exactly quantify the benefits and the disadvantages stemming from disabling the processor cache in terms of both processor failure rate and program execution time.

6 References

[1] E. Dupont, M. Nikolaidis, P. Rohr, "Embedded robustness IPs for transient-error-free ICs", IEEE Design and Test of Computers, Vol. 19,No. 3, May/June 2002, pp. 56-70

[2] S. H. Hwang, G. S. Choi, "On-chip cache memory resilience", IEEE 3rd International High-Assurance System Engineering Symposium, 1998, pp. 240-247

[3] P. P. Shirvani, E. J. McCluskey, "PADded cache: a new fault-tolerance technique for cache memories", IEEE 17th VLSI Test Symposium, 1999, pp. 440-445

[4] G. R. Brown et al., "Honeywell radiation hardened 32-bit processor central processing unit, floating point processor, and cache memory dose rate and single event effects test results", IEEE Radiation Effects Data Workshop, 1997, pp. 110-115

[5] A. K. Somani, K. S. Trivedi, "Cache error propagation model", Pacific Rim International Symposium Fault-Tolerant Systems, 1997, pp. 15-21

[6] N. S. Bowen, D. K. Pradhan, "The effect of program behavior on fault observability", IEEE Transactions on Computers, Vo. 45, No. 8 , Aug. 1996, pp. 868 -880

[7] 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

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

[9] 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

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

[11] 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

Program	Component	Wrong Answer [%]	Effect- less [%]	Latent [%]	Exception [%]	Time-out [%]	Stall [%]
MTX 4x4	D-Cache lines	2.3	97.4	0.1	0.0	0.0	0.2
	I-Cache lines	3.2	92.9	1.0	2.1	0.4	0.4
	D-Tag	2.7	92.9	0.0	0.0	0.0	4.4
	I-Tag	0.0	82.9	1.1	0.0	0.0	16.0
MTX 10x10	D-Cache lines	13.3	86.7	0.0	0.0	0.0	0.0
	I-Cache lines	3.3	93.8	0.3	2.1	0.3	0.2
	D-Tag	8.6	75.5	0.0	0.0	0.0	15.9
	I-Tag	0.0	85.1	1.2	0.0	0.0	13.7
HS 32	D-Cache lines	3.9	96.1	0.0	0.0	0.0	0.0
	I-Cache lines	2.5	89.3	0.2	3.2	2.9	1.9
	D-Tag	2.5	91.5	0.0	0.0	0.0	6.0
	I-Tag	0.0	78.2	0.8	0.0	0.1	20.9
HS 64	D-Cache lines	7.9	92.1	0.0	0.0	0.0	0.0
	I-Cache lines	2.7	89.4	0.2	3.1	3.5	1.1
	D-Tag	5.5	84.5	0.0	0.0	0.0	10.0
	I-Tag	0.0	76.2	0.8	0.0	0.1	22.9

Table 6: Analysis of SEU effects in the cache subsystem