A Low-SER Efficient Core Processor Architecture for Future Technologies

E. L. Rhod, C. A. Lisbôa, L. Carro

Universidade Federal do Rio Grande do Sul
Escola de Engenharia and Instituto de Informática
Porto Alegre, RS, Brazil
eduardo.rhod@ufrgs.br, calisboa@inf.ufrgs.br, carro@inf.ufrgs.br

Abstract

Device scaling in new and future technologies brings along severe increase in the soft error rate of circuits, for combinational and sequential logic. Although potential solutions have started to be investigated by the community, the full use of future resources in circuits tolerant to SETs, without performance, area or power penalties, is still an open research issue. This paper introduces MemProc, an embedded core processor with extra low SER sensitivity, and with no performance or area penalty when compared to its RISC counterpart. Central to the SER reduction are the use of new magnetic memories (MRAM and FRAM) and the minimization of the combinational logic area in the core. This paper shows the results of fault injection in the MemProc core processor and in a RISC machine, and compares performance and area of both approaches. Experimental results show a 29 times increase in fault tolerance, with up to 3.75 times in performance gains and 14 times less sensible area.

1. Introduction

Previously a concern only for mission critical applications, errors due to the effects of transient pulses produced by radiation and other interferences, called *soft errors*, are now being generally considered by the design community, since these errors are very likely to occur in future technologies. While successful mitigation techniques, and new memory technologies such as MRAM and FRAM, have already been devised to protect memories against soft errors, the protection of combinational logic, mainly against multiple simultaneous upsets, is a relatively recent concern and still lacks efficient solutions [1].

Due to the variability of their vulnerability periods, the SER of combinational logic is harder to quantify, and so far the mitigation of soft errors in those circuits has been dealt with through redundancy and larger transistor architectures, with obvious costs in area, power and even performance. The technology evolution towards nanoscale leads to the possibility of manufacturing chips with up to 10^{12} devices. Not only the number of transistors, but also the speed of the circuits has increased with the advent of deep sub-micron technology. All together, the result is a higher sensitivity of combinational logic to soft errors. As shown in Figure 1, from [2], while the SER of SRAM memories remains almost stable with technological scaling, the SER of logic has been always increasing.

For future technologies, solutions that impose redundancy or larger areas impair the ability to explore the advantages of the technology evolution. Therefore, new paradigms must be adopted in the design of combinational circuits to be manufactured using those technologies.

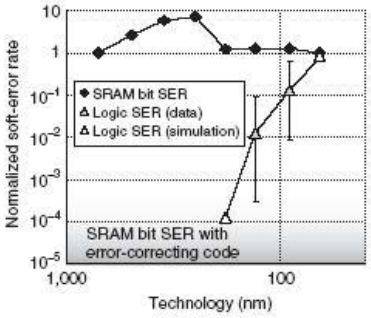


Figure 1. Evolution of SER: SRAM vs. logic [2]

Geometric regularity and the extensive use of regular fabrics is being considered as a probable solution to cope with parameter variations and improve the overall yield in manufacturing with future technologies, by using spare rows and columns that can be activated to replace defective devices. [3]. Together with the proposal of using regular fabrics, the introduction of new memory technologies that can withstand the effects of transient faults, such as

978-3-9810801-2-4/DATE07 © 2007 EDAA

ferroelectric and magnetic RAMs [2], brings back the concept of using memory to perform computations. Already proposed in the past [4], but precluded as a general purpose solution due to poor performance and high cost, the use of memory now is proposed here as a novel mitigation technique for transient faults, by reducing the area of the circuits that can be affected by soft errors.

In this paper, we try to cope with the SEU/SET problem without imposing area or performance overhead, at the same time that we favor a regular architecture that can be used to enhance yield in future manufacturing processes. We introduce a memory-based embedded core processor architecture, named MemProc, designed for use in control domain applications as an embedded microcontroller. It is a microcoded multicycle core processor that uses a reduced combinational logic and some extra memory to reduce the incidence of soft errors. Our technique reduces the area of sequential logic, which is sensible to faults, by using intrinsically protected memories.

The performance was evaluated by running in MemProc different applications selected from the targeted domain and comparing the results with those obtained using a pipelined RISC architecture.

This paper is organized as follows: section 2 discusses related work and highlights the differences between the proposed architecture and other alternatives. Section 3 describes the MemProc architecture, explaining how its simplified ALU works. Section 4 describes the fault injection process. In section 5 we comment the achieved results and also future work.

2. Related Work

The reliability of circuits manufactured in future technologies became a major topic of discussion and research in recent years [5], imposing tolerance to transient faults as a mandatory design concern.

Among different approaches to cope with soft errors found in the literature, the use of spatial or time redundancy dominates as the major technique.

The use of time redundancy to avoid undesirable errors, exploiting microarchitectural techniques that are already incorporated in the processor due to performance reasons, has been proposed in [6], and a penalty of up to 30% in performance is incurred. The use of simultaneous multithreading to detect transient faults is also proposed in [7]. The area cost of such duplication techniques is obviously high.

In [8], a self-repairing unit for microprogrammed processors is proposed. In that work, the authors used a dedicated built-in self-test (BIST) architecture to provide an online status – either good or faulty – for each block in the execution unit. For each processor microinstruction, they defined a sequence of microinstructions that can execute the same operation using only fault-free units. This approach has a significant area and performance overhead due to the BIST and fault-free units added to the circuit.

In [9], the authors propose the use of a self-stabilizing microprocessor to cope with any combination of soft errors. The paper presents only the initial studies of the behavior of the self-stabilizing processor in the presence of soft errors. Whenever affected by a transient fault, the processor is able to converge to a safe state, from which the normal fetch-decode-execute sequence can be resumed during fault-free periods. Besides presenting the design scheme for the processor, a new technique for the analysis of the effects of soft errors is introduced, which instead of using simulation is based in an upper bound algorithm that does not take into account the fault masking effects of the circuit.

The use of memory as a computing device, has been subject of research in the past. In order to explore the large internal memory bandwidth, designers proposed to bring some functions executed by the processor into memory [4]. This technique apparently has been discarded due to its limited field of application.

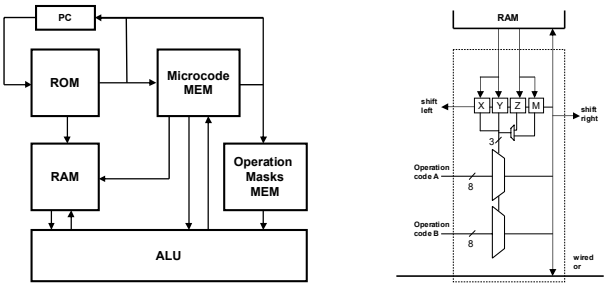
Back to the fault tolerance arena, another strong argument to the use of memory to perform computation functions is its intrinsic protection against defects, due to the use of spare columns and spare rows, such as in DRAMs. More recently, the fact that new memory technologies, such as ferroelectric RAMs (FRAMs), magnetic RAMs (MRAMs), and flash memories, are virtually immune to soft errors, due to their physical characteristics [2], makes those types of memories an important additional resource for the implementation of fault tolerant systems. MRAMs are also more energy efficient than other non-volatile memory technologies, since they consume less power during read and write operations [10]. Since memories are regular structures by nature, memory systems will also benefit from the foreseen advantages that regular fabrics will provide for future technology.

The fact that the proposed MemProc processor relies heavily in the use of memories adds the benefits arising from regularity and immunity against soft errors to the solution proposed in this paper. In order to highlight the fault tolerance of the design, we injected faults and compared the results with those obtained for another core processor (MIPS), using the same simulation tool. In this process, two implementations of each architecture running in parallel have been simulated and faults have been injected in one of them, comparing the produced results for each possible single event transient occurrence. Therefore, all the possible fault incidence cases have been considered, even those in which the faults are masked by the architecture and do not generate errors.

3. The Architecture of MemProc

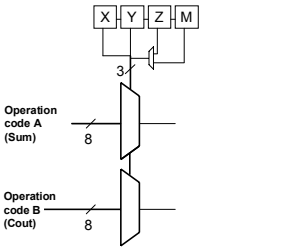
The architecture proposed in this paper is a microcoded multicycle 16-bit core processor with Harvard architecture, in which part of the datapath has been replaced with memory, thereby reducing the amount of combinational logic. In Figure 2(a) the main functional blocks of the proposed architecture are shown.

The application code, which is also called the *macroinstrucion code*, is stored in the ROM memory. The instructions in this code, as usual, indicate the operations to be performed and their operands. The *microcode memory* receives the initial microcode address of the current operation from the ROM memory, and generates the control signals for the data memory, ALU and operation masks memory. The operation masks memory is responsible for passing the operation masks to the ALU. All arithmetic and logic operations results are stored in the RAM memory, and the register bank is also mapped into this memory.



2-bit addition:
Truth table

A	B	Cin	Sum	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1



(c)

Figure 2. (a) MemProc architecture (b) ALU for one bit operation (c) 2-bit addition using MemProc ALU

In the MemProc ALU, operations are performed by 8:1 multiplexors, which are able to generate all the minterms for a given 3-bit boolean function, according to the values of bits X, Y, and Z (or M). Figure 2(b) depicts a MemProc ALU block for processing 1-bit operands.

The complete MemProc ALU is 16-bit wide and their 16 blocks work in parallel, being able to perform bit serial arithmetic and logic operations. All operation mask values are independent from each other, so each processing element of the ALU can perform a different Boolean function. To accelerate addition operations, we use two 8:1 multiplexors instead of a single one; one multiplexor is used to calculate the sum and the other to calculate the carry out. An extra flip-flop, called "M", was also added, to accelerate multiplications.

In figure 2(c) the addition of two 1-bit operands is used to illustrate how the ALU works. We can see from the truth table the operation masks for the "sum" and the "cout" (carry out) outputs of the multiplexors. Also in figure 2(c), we can se the presence of a wired-or bus. This bus implements an "or" operation of all the multiplexors' outputs. This wired-or bus is an extremely important element in what we call "compute only the necessary to get the result", which will be discussed in the following paragraph.

The way MemProc achieves its high performance is based on the fact that it computes just the necessary cycles to get the operation result. In traditional computer architectures, the ALU does its arithmetic and logic operations using combinational hardware that always takes the same time to compute the operation, regardless of the value of the operands. MemProc executes only the number of cycles required to get the result, depending on the carry propagation chain.

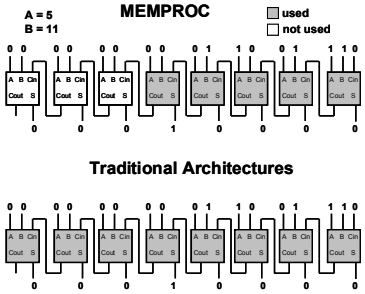


Figure 3. 8-bit Addition paradigm.

In Figure 3 we can see that MemProc requires only 5 of the 8 operation units to perform the addition of two 8-bit operands, which means that it takes 5/8 of the time required by traditional architectures to perform this operation. To detect when the operation is finished, MemProc uses the wired-or bus to evaluate when there are no more carry-outs to propagate, which means that the addition has finished. This way, we can say that the proposed architecture takes advantage on the value of the operands. For instance, one addition can require from 3 to 18 cycles to be performed, depending on the number of carries to be propagated. On the other hand, store operations require only 2 cycles.

In multiplications, the number of cycles depends on the number of bits equal to zero in the operands. The number of required cycles decreases as the number of bits equal to zero in the operands increases. One could say that if the values of the operands are high the proposed approach would not have any advantage. However, as shown in [11], the

transition activity for some multimedia benchmarks is more intense in the 8 least significant bits.

4. Experimental Results: fault tolerance, area and performance metrics

The fault rate of a circuit, also known as soft error rate, can be expressed by the amount of errors that affect the circuit in a certain period of time.

The soft-error rate of a design can also be expressed by the nominal soft-error rate of the individual circuit elements that compose the design, like memory structures such as SRAMs, sequential elements such as flip-flops and latches, combinational logic and its architectural and timing vulnerability characteristics [12] as follows:

$$SER^{design} = \sum_{i} SER_i^{\text{nominal}} \times TVF_i \times AVF_i \quad (2)$$

where i represents the i^{th} element of the design.

The $SER^{nominal}$ for the i^{th} element is defined as the soft failure rate of a circuit or node under static conditions, assuming that all the inputs and outputs are driven by a constant voltage. The TVF_i, time vulnerability factor (also known as time derating) stands for the fraction of the time that the element is susceptible to SEUs, which will cause an error in the i^{th} element. The AVF_i, architectural vulnerability factor (also known as logic derating) represents the probability that and error in the i^{th} element will cause a system-level error. In this study the time vulnerability factor was not taken into account [12].

One of the most usual ways to measure the SER of a circuit is evaluating the number of Failures in Time (FIT), which means one error every 10^9 hours [12, 13]. A soft error rate of 10 FIT means that the device will generate 10 errors in 1 million years. Another commonly used metric to express SER is the Mean Time to Failure (MTTF). As an example, a MTTF of 1000 hours means that, in average, one error occurs after 1000 hours of device operation. FIT and MTTF are inversely related, i.e., less FIT means better SER, while higher MTTF means better SER [13]. In this paper we use the MTTF metric to measure the fault tolerance of the proposed architecture and MIPS.

In order to evaluate the feasibility of the architecture proposed in this paper, both in terms of fault tolerance, area, and performance, extensive simulations have been executed, using an in-house developed simulation tool named CACO-PS (a System C-like simulator) [14]. The comparisons have been made against the well-known MIPS 16-bit RISC architecture, with a 5-stage pipeline and forwarding unit [15], widely used in real-world embedded processors.

4.1 Fault Rate Evaluation

To evaluate the fault rate of the processors, random faults were injected in both MIPS and MemProc during their operation. During fault injection, the behavior of each processor was compared to the behavior of its fault free version when executing the same application with the same data.

Since some faults may hit parts of the circuit which are not being used at a specific moment in time, to detect if a fault has been propagated or not it is not necessary to compare the value of all functional units or registers. It is only necessary to compare those components that are vital for the correct operation of the system. For the MIPS processor, the units to be checked are the program counter, in order to detect wrong branches, and wrong data or address values during write operations, to identify silent data corruption (SDC). In the case of MemProc, besides the program counter, the microcode counter was checked to identify wrong branches, and the write address and write data contents were checked to identify SDC. The CACO-PS tool has also been used to implement the fault injection and detection circuits.

It is clear that the probability of a component being hit by a fault increases with the area of the component. So, to be as realistic as possible, we have implemented the random fault injector following this probabilistic fault behavior. To do so, we have created a file with all the important information about the components, such as component size, number of outputs and outputs widths. Then, when the fault injection process starts, this component information file is loaded by the random fault injector and is used to determine which is the component that fails in each fault injection cycle, according to a probability based on its area.

Another important variable in the fault injection process is the amount of faults that are injected in every cycle. In this work, we decided to use a technique called environmental acceleration [16], otherwise, we would have to wait for long simulation times in order to get an error. To make calculations easier, we assumed that the particle flow is able to produce 1 SEU or SET every 2 cycles in the MIPS processor, which is indeed a high rate assumption. To calculate the corresponding number of faults per cycle for the MemProc processor, we have used the Leonardo Spectrum tool [17] to generate area and timing information from the VHDL descriptions of both processors. Table 1 shows those results and the corresponding time gap between faults for both processors. As one can see, this time gap is inversely proportional to the number of SET sensible gates of the processors, showing that the lower the sensible area of a circuit, the lower is the probability of a particle hit affecting that circuit.

As shown in Table 1, the combinational circuit (# of sensible gates) in the MemProc architecture is very small when compared to the size of its memory elements. In our approach, the memory elements are considered to be immune to soft errors, since we are supposing the use of new memory technologies, such

as MRAM, FRAM, and flash memories, as mentioned before. In order to allow a fair comparison, during the fault injection process all memory elements of both architectures have been considered immune to soft errors.

Table 1. Area and number of faults per cycle.

Architecture	MemProc	MIPS
ROM (bits)	1,792	2,720
RAM (bits)	512	512
Op. Masks Mem. (bits)	128 x 256	-x-
Microcode Mem. (bits)	1024 x 68	-x-
# of sensible gates	679	9,619
Frequency (MHz)	254	54
time between faults (ns)	523.62	37.04

The fault injection process injected random faults according to the probability of the component being hit and also the calculated number of faults per cycle. In this process, faults were injected until one error or a silent data corruption (SDC) was detected, in order to determine the time to failure. This process was repeated 100 times, and the mean time to failure calculated as the average time to failure in the 100 experiments. The results are shown in Table 2, which lists the fault injection results for the MemProc and MIPS processors.

Table 2. Fault rates for both architectures.

Architecture	MemProc	MIPS
# of cycles	585,945	4,320
# of injected faults	4,404	2,160
# of errors + SDCs	100	100
MTTF (μs)	23.068	0.798

The first line of Table 2 shows the number of cycles each processor had to execute until 100 errors or SDCs were detected. The second line presents the number of faults injected during the process. The third line shows the total number of errors and SDCs that occurred during this process. The fourth line shows the corresponding Mean Time To Failure value, showing that the MTTF of the MemProc architecture is almost 29 times bigger than the MIPS's one. These results show the significant reduction in the MTTF that can be obtained by using the proposed architecture.

4.2 Performance Evaluation

The performance of MemProc was evaluated using a cycle accurate simulation tool (CACO-PS [14]) to measure the number of cycles required to execute a set of benchmarks. Using the description language of CACO-PS, which is similar to System C, both MemProc and MIPS architectures were described and simulated. The performance evaluation was done using four different application programs, with different processing characteristics: three sort algorithms and the IMDCT (Inverse Modified Discrete Cosine Transform, part of the MP3 coding/decoding algorithm) function, which were executed both in MemProc and MIPS. Those applications have been selected because are widely used in the target domain (control applications), and also because they use most of the operations implemented in the MemProc instruction set.

The maximum frequency of operation for both architectures was evaluated using VHDL descriptions, and the Leonardo Spectrum tool. The obtained results are shown it Table 3, in which we can see that MemProc executes the bubble sort algorithm in approximately 4.7 thousand cycles, while MIPS takes half this number of cycles to perform the same. As stated before, MemProc requires several cycles to perform arithmetic (bit serial) operations, and the number of cycles also depends on the value of the operands. That is the reason why the number of cycles spent by MemProc is higher than that of MIPS. On the other hand, the critical path of MemProc is determined by the access time of the microcode memory, while in MIPS the critical path is determined by the multiplier delay. So, the maximum frequency of MemProc is more than 4 times higher than that of MIPS, and, as consequence, the MemProc is almost 3 times faster than MIPS running the sort algorithms.

Table 3. Performance when executing benchmark applications

Application	MIPS (54 MHz)	
	# of Cycles	Computation Time (μs)
Bubble Sort	2,280	42.2
Insert Sort	1,905	35.3
Select Sort	1,968	36.4
IMDCT	38,786	718.3

Application	MemProc (254 MHz)		Perform. Ratio
	# of Cycles	Computation Time (μs)	
Bubble Sort	4,720	18.4	2.29
Insert Sort	2,508	9.8	3.60
Select Sort	2,501	9.7	3.75
IMDCT	142,961	562.8	1.28

The analysis of the results when executing IMDCT shows that MemProc was only 1.28 times faster. That happens because this algorithm uses the multiply instruction, which can take up to 48 cycles to be executed in MemProc. It is important to mention here that MemProc is a multicycle machine, while MIPS is a pipelined one, which is expected to be faster than its multicycle version. So, we can conclude that if we were comparing MemProc with MIPS multicycle version, performance results would be even better. The performance gains of MemProc come from the fact that the number of cycles it takes to perform an operation depends both on the operation and on the values of the operands. For instance, let us consider that MIPS needs 1 cycle to perform an add operation. Since the frequency of MemProc is almost 5 times higher, if the operands are

such that the number of carry cycles are less than 5, MemProc will finish the addition operation earlier than MIPS. Also, store operations take only 2 cycles in MemProc, which is more than 2 times faster than in MIPS.

In order to stress that the primary goal of the proposed technique is fault tolerance, and not performance, the execution of the IMDCT application has been executed once again, this time with MemProc running at 198.44 MHz, which gives the same computation time for both MIPS and MemProc running that application. The fault injection process was then repeated for MemProc running at that frequency and the MTTF has been recomputed. The resulting MTTF was 18.7 μs, which is still more than 23 times longer than that of MIPS. This confirms our claim, that the approach proposed in this paper is well suited to a higher reliability embedded processor.

5. Conclusions and Future Work

This work proposes a novel fault tolerant architecture for embedded core processors for use in control applications, which uses microcoded memory to execute macroinstructions, and uses as ALU sixteen 8:1 multiplexors to perform all logic and arithmetic operations. Simulation results have shown that the Mean Time to Failure of the proposed architecture is more than 29 times longer than the MIPS one, due to the reduction of the area sensitive to faults, without having any performance degradation, on the contrary, with improved performance. Also, results showed that, despite requiring several cycles to execute its bit serial operations, MemProc was 1.28 times faster than MIPS. While the main goal of this work was to propose a new fault tolerant architecture, the performance gains come from the fact that MemProc exploits the benefits of using bit serial operations, and differently from MIPS, it can require less cycles to make the same operation, depending on the value of the operands.

The proposed architecture, while not being a final solution, reflects our focus in the search for new processor design alternatives that might be used in the future, when current ones will start to fail due to the weaknesses of new technologies. It innovates in several design features, even providing better performance when compared to a well known architecture for embedded applications, while providing much more reliability against transient faults. In order to stress that fault tolerance is the major goal of this work, a lower frequency version of MemProc, which delivers exactly the same performance of the alternative architecture for the sample application, has been used in one experiment.

6. References

[1] Rossi, D., Omaña, M., Toma, F. and Metra, C., "Multiple Transient Faults in Logic: An Issue for Next Generation ICs ?", in Proceedings of th 20th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT 2005), pp. 352-360, IEEE Computer Society, Los Alamitos, CA, Oct '05.

[2] Baumann, R., "Soft Errors in Advanced Computer Systems", IEEE Design and Test of Computers, vol. 22, no. 3, pp 258-266, IEEE Computer Society, May-June 2005.

[3] Sherlekar, D., "Design Considerations for Regular Fabrics", in Proceedings of the 2004 International Symposium on Physical Design (ISPD 2004), pp. 97-102.

[4] Elliott, D.G., Stumm, M., Snelgrove, W.M., Cojocaru, C., Mckenzie, R., "Computational RAM: implementing processors in memory", Design & Test of Computers, IEEE, vol. 16, no. 1, pp. 32-41, IEEE Computer Society, Jan/Mar 1999.

[5] Semiconductor Industry Association. International Technology Roadmap for Semiconductors – ITRS 2005, last access July, 2006. http://www.itrs.net/Common/2005ITRS/Home2005.htm.

[6] Rotenberg, E., "AR-SMT: A Microarchitectural Approach to Fault Tolerance in Microprocessors," in Digest of Papers of the 29th Annual International Symposium on Fault-Tolerant Computing, pp. 84-91, IEEE Computer Society, New York-London, 1999, ISBN: 0-7695-0213-X.

[7] Reinhardt, S. K., and Mukherjee, S. S., "Transient Fault Detection via Simultaneous Multithreading," in Proceedings of the 27th Annual International Symposium on Computer Architecture (ISCA 2000), pp. 25-36, ACM Press, May 2000.

[8] Benso, A.; Chiusano, S.; Prinetto, P.,"A self-repairing execution unit for microprogrammed processors", in IEEE Micro, vol. 21, issue 5, pp. 16-22, IEEE Computer Society, New York-London, sept-oct 2001.

[9] Dolev, S.; Haviv, Y.A., "Self-Stabilizing Microprocessor: Analyzing and Overcoming Soft Errors" in IEEE Transactions on Computers, vol. 55, no. 4, pp. 385-399, IEEE Computer Society, New York-London, April 2006, ISSN: 0018-9340.

[10] Tehrani, S. et al., "Magnetoresistive Random Access Memory using Magnetic Tunnel Junctions", in Proceedings of the IEEE, vol. 91, no. 5, pp 703-714, IEEE Computer Society, London-New York, May 2003. ISSN: 0018-9219.

[11] Ramprasad, S., Shanbhag, N. R., Hajj, I. N., "Analytical Estimation of Transition Activity from Word-level Signal Statistics", in Proc. of the 34th Design Automation Conference (DAC'97), pp. 582-587, IEEE Comp. Soc., June 1997.

[12] N. Seifert and N. Tam, "Timing Vulnerability Factors of Sequentials," IEEE Trans. Device and Materials Reliability, V4, N3, Sept. 2004, pp. 516-522.

[13] S. S. Mukherjee, J. Emer, and S. K. Reinhardt, "The Soft Error Problem: An Architectural Perspective," in Proceedings of the 11th International Symposium on High-Performance Computer Architecture (HPCA), pp. 243-247, IEEE Computer Society, Los Alamitos, CA, Feb. 2005, San Francisco.

[14] A. C. S. Beck Fº, J. C. B. Mattos, F. R. Wagner, and L. Carro, "CACO-PS: A General Purpose Cycle-Accurate Configurable Power-Simulator", in Proceedings of the 16th Brazilian Symposium on Integrated Circuits and Systems Design (SBCCI 2003), Sep. 2003.

[15] Patterson, D.A., and Hennessy, J. L.. Computer Architecture: a Quantitative Approach, 3rd Edition, Elsevier Science & Technology Books, June 2002. ISBN: 1558605967.

[16] Mitra, S., Seifert, N., Zhang, M., Shi, Q., Kim, K.S., "Robust system design with built-in soft-error resilience", in Computer, vol. 38, issue 2 pp. 43-52, feb 2005.

[17] Last access: July, 2006. http://www.mentor.com/products/fpga_pld/synthesis/leonardo_spectrum/.