A Novel Co-Design Approach for Soft Errors Mitigation in Embedded Systems

F. Restrepo-Calle, A. Martínez-Álvarez, F. R. Palomo, H. Guzmán-Miranda, M. A. Aguirre and S. Cuenca-Asensi

***Abstract*—A novel proposal to design radiation-tolerant embedded systems combining hardware and software mitigation techniques is presented. Two suites of tools are developed to automatically apply the techniques and to facilitate the trade-offs analyses.**

***Index Terms*—Fault tolerance, Radiation effects, Reliability.**

I. Introduction

During last decades the progressive miniaturization of electronic components has led important advances in microprocessors, such as the dramatically increase of their performance. However, while technology shrinks, voltage source level and noise margins are also reduced, causing that electronic devices have become less reliable and microprocessors more susceptible to *transient faults* induced by radiation. These intermittent faults do not provoke a permanent damage, but may result in an incorrect program execution by altering signal transfers or stored values, the so called *soft errors* [1].

In order to face reliability problems, applying redundant hardware has been the usual solution. From low level structures, using techniques like: Error-Correcting Code, parity bits, Triple Modular Redundancy (*TMR*); up to more complex components like functional units [2], co-processors [3]; or by means of exploiting the multiplicity of hardware blocks available on multi-threaded/multi-core architectures [4], [5]. Most hardware approaches provide a high accurate solution for transient faults. However, these techniques are unfeasible in many cases due to high costs involved.

Motivated by the need of low cost solutions, several proposals based on redundant software have been developed [6], [7], [8]. While software-based approaches are cheaper than hardware-based ones, they cannot achieve the same performance or reliability, since they must execute additional instructions.

Manuscript received April 2, 2010.

This work makes part of *RENASER* project (ESP2007-65914-C03-03) funded by the 2007 Spain Research National Plan of the Ministry of Science and Education in which context this work has been possible. The work presented here has been carried out thanks to the support of the research project 'Aceleración de algoritmos industriales y de seguridad en entornos críticos mediante hardware' (GV/2009/098) (Generalitat Valenciana, Spain).

F. Restrepo-Calle, A. Martínez-Álvarez and S. Cuenca-Asensi are with the Computer Technology Department, University of Alicante, Carretera San Vicente del Raspeig s/n, 03690 Alicante, Spain.

F. R. Palomo, H. Guzmán-Miranda and M. A. Aguirre are with the Department of Electrical Engineering, University of Sevilla, Camino de los Descubrimientos, 41092 Sevilla, Spain.

In the case of embedded systems, there are large domains of applications where factors like cost, power and performance are as important as reliability. For this kind of applications, optimal solutions can be found combining software and hardware aspects of different techniques. Several of these hybrid approaches has been proposed in recent years showing promising results [9], [10]. However, they are very specific and lack of flexibility to get the best trade-offs between fault detection capabilities and the usual embedded design constraints.

In this context, the first contribution of this work is the use of the co-design methodology [11] in the development of hybrid soft errors mitigation strategies. That is, the process of exploring the space of hardware and software techniques to achieve a customized fault tolerant version of the system that best meet the requirements of the application. To aim this goal, as the second contribution of this paper, two suites of tools have been developed: a *Software Hardening Environment* [12] aimed to implement, automatically apply and evaluate software-only fault tolerant techniques; and a *FPGA*-based fault emulation tool, called *FTUnshades* [13], that permits to assess several reliability metrics of the overall embedded system. As case study for validating our approach, we have explored the hardening of an embedded application based on the *PicoBlaze* soft-microprocessor [14]. Mitigation techniques are applied to a high abstraction level so the final deployment platform could be an *ASIC* or an anti-fuse *FPGA*. Different trade-offs among code overhead, performance, fault coverage and hardware costs have been studied.

II. Hardening Infrastructure

A. Software Hardening Environment

This environment lets the user to design and implement software-only fault tolerance techniques, and also to automatically apply them into programs. It is made up of a generic *Hardener*, and a generic *Instruction Set Simulator—ISS*, jointly with several compiler front-ends and back-ends to deal with different microprocessor targets (Fig. 1).

A given compiler front-end takes the original source code from a supported target, performs lexical, syntactical and semantic analyses, and finally generates the *Generic Instruction Flow* (*GIF*) as output. This flow represents an intermediate high level abstraction of the program. Then the hardening tasks are performed within the *Generic Hardening Core* (*GH-Core*). Finally, it produces the *Hardened-GIF* (*HGIF*) which is then re-targeted to a custom supported microprocessor. In this way,

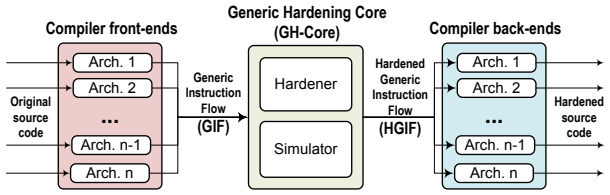

Fig. 1. Software Hardening Environment

it is also possible to generate protected code for a different target than original.

Among various advantages of our proposal we highlight that the *GH-Core* is based on a *Microprocessor Generic Architecture*. It allows providing an uniform hardening environment that permits to design and implement different techniques in a platform independent way. The automatic generation of hardened code is guided by instruction-level code transformation rules (we obtain hardened assembler code).

The *GH-Core* has two main components: the *Hardener* and the *ISS*. The *Hardener* comprises a sort of common procedures that can be used to design new hardening software-based techniques. On the other hand, the *ISS* assists the designer in the implementation of them. It allows to perform different analyses on the *GIF* and *HGIF* to check the correctness of the hardening process, and also offers useful information in the co-design process: time and space overheads, fault-coverage estimations, etc. In order to evaluate the reliability provided by the software techniques, the *ISS* is also able to simulate *Single Event Upset—SEU* faults by means of a single *bit-flip* into a bit. Therefore, the *Hardener+ISS* offer a rich set of co-design parameters that can be used to perform an exhaustive soft-wide design space exploration.

Considering hardening purposes, as it was suggested by Reis et al. [8], the *Hardener* classifies in a special way those instructions whose function imply to cross the borders of the *Sphere of Replication* (*SoR*) [15]. The *SoR* is the logic domain of redundant execution. Therefore, when an instruction causes that some data enter inside the *SoR* (e.g. reading a port, loading a value into a register or reading from memory), we will classify it as *inSoR*; and consequently when an instruction provokes data going out from the *SoR* (e.g. writing on a port, storing into the memory), we will classify it as *outSoR*. Note that the boundaries of the *SoR*, and consequently the coverage of the protection, could change according to the implemented technique. For instance, including/excluding the memory subsystem inside the *SoR* jointly with the register file, or even moving some selectively-chosen registers from the register file inside/outside the *SoR*.

B. SEU-Emulation Tool: FTUnshades

The second main component of the hardening infrastructure is *FTUnshades* [13]. It is a *FPGA*-based platform for the study of digital circuit reliability against radiation-induced soft errors. *SEU* affecting the circuit are emulated by inducing bit-flips in the circuit under study, by means of dynamic partial reconfiguration. The system is composed of a *FPGA* emulation board and a suite of software tools for testing the emulated design and analyzing test results. In this work, *FTUnshades* is used to assess the reliability of the full *HW/SW* mitigation strategy applied in the physical implementation of the system.

III. Experiments and Results

In order to validate our proposal, we have designed and evaluated a number of radiation-tolerant versions of an embedded system. The co-design space exploration is driven by a well known application (*mmult* - matrix multiplication). The hardware is composed of a technology-independent (i.e. valid for *ASIC* or *FPGA*) version of *PicoBlaze* developed for this work (*RTL PicoBlaze*). This micro is a 8-bit width softcore widely used within *FPGA*-based applications. We have had two sources of variation when tuning our system, the different hardened versions of the software, and the selective hardening of the *RTL PicoBlaze*. Both of them are controlled by a sort of design parameters of interest (set of registers to harden by software or hardware, fault tolerance technique, etc.).

A. Development of hybrid hardening strategies

In this case study, an adaptation of *SWIFT-R* [16] has been implemented. This is a software-only recovery technique that consists of triplication of data and instructions, jointly with the insertion of verification points to check data consistency (by means of majority voters).

The flexibility of our *Software Hardening Environment* allows to apply automatically specific techniques in a selective way. In this case study, *SWIFT-R* has been implemented to be incrementally applied to several subsets of selected registers from the microprocessor register file. This is possible by moving the remaining registers outside of the *SoR*. In addition, to help the designer prioritizing registers to protect, the *GH-Core* gathers information about: the number of clock cycles the microprocessor registers must keep a valid value (*lifetime*, which has a high impact on reliability), and the code and execution time overheads. Fig. 2 shows the overheads results for several selectively hardened registers subsets (from sixteen possible *PicoBlaze* registers, hexadecimal numbered). These results are normalized with a baseline built with the non-hardened version. The registers used in the *mmult* program are: *0*, *A*, *D*, *E*, and *F*. Those ones with the higher *lifetime* are A and F (85.7% and 83.3% of the total clock cycles, respectively), and consequently they are the firsts candidates to be hardened.

On the hardware side, the fault tolerant co-design strategy was complemented by incrementally hardening some microprocessor resources. It was done by manually applying *TMR* to different subsets of micro-architectural registers. The following versions have been generated: non-hardened *RTL PicoBlaze* (*P0*); microprocessor with hardware redundancy for Program Counter – *PC*, Flags and Stack Pointer – *SP* (*P1*); hardware redundancy for all registers in the pipeline (*P2*); hardware redundancy for *PC*, Flags, *SP*, and Pipeline (*P3*); and

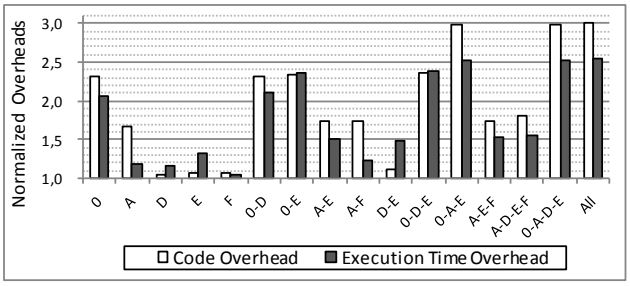

Fig. 2. Normalized code and execution time overheads

a full protected version, i.e. microprocessor with redundancy for Register file, *PC*, Flags, *SP*, and Pipeline (*P4*).

Using the information provided by the *GH-Core* and the synthesis tools, the designer can select the bests candidates for further analyses. However, for demonstration purposes, all the systems, 16 software versions jointly with the 5 different version of the microprocessor (in total 80 systems), have been synthesized and implemented using the *Xilinx ISE* 10.1 suite tool.

B. Reliability evaluation

To evaluate reliability, the well known *SEU* fault model is used. That is, only one *SEU* is injected during each program execution. The fault is injected by a bit-flip in a randomly selected bit from the microprocessor during a program execution. Faults were classified according to their effect on the expected program behavior as in [16]. If a fault makes the program to finish without producing the expected output, it is marked as a *Silent Data Corruption* (*SDC*) fault. If the program finishes producing the expected output, the fault is marked as an *unnecessary for Architecturally Correct Execution* (*unACE*). Finally, if a fault causes the abnormal program termination or an infinite execution loop, fault is marked as a *Hang*.

For calibration purposes of *FTUnshades* tool, this is, calculate the minimal number of *SEUs* needed for an accurate fault emulation campaign, a previous experiment was carried out. In this way, Fig. 3 presents the *unACE* faults percentage obtained varying the number of injected *SEUs* to the register file of the *P0* micro running the non-hardened program, which is the worst case scenario. The results show that the 95% confidence interval is less than $\pm 1.0\%$ after 5200 injected *SEUs*.

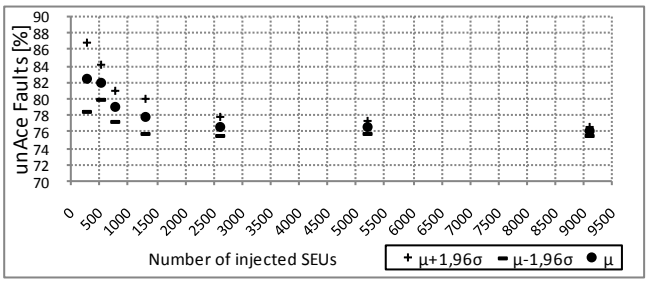

Fig. 3. Incremental injection fault campaign for calibration purposes

A second experiment using *FTUnshades* was performed to evaluate the overall reliability of every one of the 80 configurations of the system. Every test campaign makes selective attacks on the microprocessor register subsets (including register file, *PC*, flags, *SP* and pipeline). In each register subset, 5200 *SEUs* (one per run) have been emulated in a randomly selected clock cycle from all the workload duration, for a total of 26000 *SEUs*. Fig 4 presents the fault classification percentages obtained for each system. These results are the weighted average of the results from the selective attacks to the internal microprocessor register subsets, assuming the same fault probability for all bits on target.

Note that the *SWIFT-R* technique offers a considerable reliability increment, even in the non-hardened hardware (up to 95.38% *unACE* faults in the full hardened program), which is much higher than the reliability of every hardware-hardened approach using the non-hardened program. Results for the *P4* micro approach are not showed in Fig. 4 because 100% of the injected faults were classified as *unACE*, as expected. Furthermore, notice that combining *SWIFT-R* with hardware protection only to critical registers, such as *PC*, Flags, and *SP* (*P1* approach), gives a remarkable reliability increase (up to 98.32% *unACE* faults).

C. Discussion

Taking into account the requirements of the application under design, the analysis of overheads jointly with reliability results is a very important key during the co-design process. These results facilitate to find the solutions having the best robustness/overhead compromise. For instance, *SWIFT-R* applied only for the register subset $\{A, D, E, F\}$ running in the *P3* microprocessor is an interesting choice, because it offers both, high reliability (98.68% of *unACE* faults), and acceptable code and execution time overheads ($\times 1.80$ and $\times 1.56$ respectively).

Although reliability is higher when combining software-hardened programs with hardware-redundant approaches (for instance, up to 99.10% *unACE* faults for the *P3* micro), hardware costs are also higher, which is an important fact that must be considered. Fig 5, on the one hand, shows the hardware costs of each approach normalized with a baseline built with the non-hardened *RTL PicoBlaze* (*P0*); on the other hand, it also depicts, in a secondary axis, the percentage of *unACE* faults obtained for the *SWIFT-R* program (*all* registers hardened). This figure permits to see at a glance, how reliability and costs are affected by every studied hardware approach. The hardware costs were provided by the *Xilinx XST 10.1* synthesis tool. They are expressed in terms of: Flip/Flops and Latches, primitives (mux, luts, etc.), and RAM (distributed and block ram).

It is worth noting that hardware cost increases considerably when the registers in the pipeline are hardened (*P2* and *P3*), whereas the reliability only improves slightly in these cases, or even decreases if compared with cheaper approaches (*P2+SWIFT-R*). In case of *P4*, high hardware costs may result unsuitable for many designs, although its reliability is 100%.

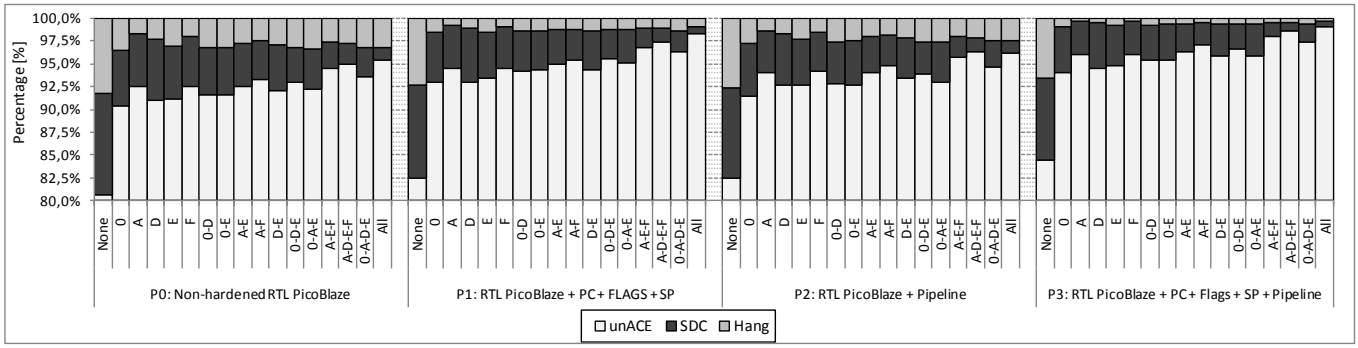

Fig. 4. Fault classification percentages for every selectively-hardened software version and selectively-hardened hardware approach

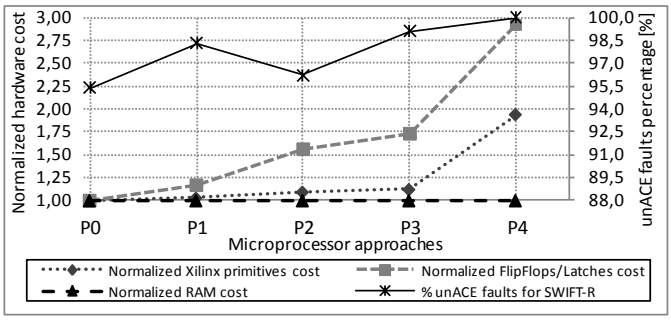

Fig. 5. Normalized hardware cost and percentage of *unACE* faults per every microprocessor version

Finally, obtained results allows the designer to decide which *HW/SW* configuration best meet the requirements of each specific application. For instance, in this case study might result as a suitable configuration the prototype with the *P1* hardware and *SWIFT-R* applied to the register set $\{A\text{-}D\text{-}E\text{-}F\}$ on the software side, because it offers high level of reliability (97.43% of *unACE* faults) with acceptable costs and overheads. In some other applications, for example, if the performance degradation exceeds required, it might be preferable applying another more lightweight technique for the software, and incrementing protection and cost on the hardware side.

IV. Conclusions and Future Work

This paper presents a novel approach to design fault tolerant embedded systems. The co-design methodology jointly with the developed tool suites allow an easy exploration of the design space between hardware-only and software-only mitigation techniques. The resulting hybrid strategies are tuned to best fit the reliability requirements and the design constraints at the same time. The advantages of our proposal have been illustrated by means of a case study. As future work, the infrastructure can be extended to support 32-bit microprocessors, given the fact that it is based on a *Microprocessor Generic Architecture*.

References

[1] R. Baumann, "Radiation-induced soft errors in advanced semiconductor technologies," *IEEE Trans. on Device and Materials Reliability*, vol. 5, no. 3, pp. 305–316, Sept 2005.

[2] T. Austin, "DIVA: A reliable substrate for deep submicron microarchitecture design," in *32nd Annual Int. Symp. on Microarchitecture, (MICRO-32)*, 1999, pp. 196–207, Haifa, Israel, Nov 16-18, 1999.

[3] A. Mahmood and E. McCluskey, "Concurrent error-detection using watchdog processors," *IEEE Trans. Comput.*, vol. 37, no. 2, pp. 160–174, Feb 1988.

[4] S. Mukherjee, M. Kontz, and S. Reinhardt, "Detailed design and evaluation of Redundant Multithreading alternatives," in *29th Int. Symp. on Computer Architecture*, 2002, pp. 99–110, AK, May 25-29, 2002.

[5] M. Gomaa, C. Scarbrough, T. Vjaykumar, and I. Pomeranz, "Transient-fault recovery for chip multiprocessors," *IEEE MICRO*, vol. 23, no. 6, pp. 76–83, Nov-Dec 2003.

[6] B. Nicolescu, Y. Savaria, and R. Velazco, "Software detection mechanisms providing full coverage against single bit-flip faults," *Ieee Transactions on Nuclear Science*, vol. 51, no. 6, pp. 3510–3518, 2004.

[7] N. Oh, P. P. Shirvani, and E. J. McCluskey, "Error detection by duplicated instructions in super-scalar processors," *IEEE Transactions on Reliability*, vol. 51, no. 1, 2002.

[8] G. A. Reis, J. Chang, N. Vachharajani, R. Rangan, and D. I. August, "SWIFT: software implemented fault tolerance," *CGO 2005: Int Symposium on Code Generation and Optimization*, pp. 243–254, 2005.

[9] G. Reis, J. Chang, N. Vachharajani, S. Mukherjee, R. Rangan, and D. August, "Design and evaluation of hybrid fault-detection systems," in *32nd International Symposium on Computer Architecture, Proceedings*, 2005, pp. 148–159, madison, WI, Jun 04-08, 2005.

[10] P. Bernardi, L. Bolzani, M. Rebaudengo, M. Reorda, F. Vargas, and M. Violante, "A new hybrid fault detection technique for systems-on-a-chip," *IEEE Trans. Comput.*, vol. 55, no. 2, pp. 185–198, Feb 2006.

[11] G. DeMicheli and R. Gupta, "Hardware/software co-design," *Proc. of the IEEE*, vol. 85, no. 3, pp. 349–365, MAR 1997.

[12] F. Restrepo-Calle, A. Martínez-Álvarez, S. Cuenca-Asensi, F. Palomo, and M. Aguirre, "Hardening development environment for embedded systems," 2010, in the 2nd HiPEAC Workshop on Design for Reliability (DFR'10) held in conjunction with The 5th Int. Conf. on High Performance and Embedded Architectures and Compilers. Pisa, Italy, Jan 25-27, 2010.

[13] H. Guzman-Miranda, M. Aguirre, and J. Tombs, "Noninvasive fault classification, robustness and recovery time measurement in microprocessor-type architectures subjected to radiation-induced errors," *IEEE Transactions on Instrumentation and Measurement*, vol. 58, no. 5, May 2009.

[14] K. Chapman, *PicoBlaze KCPSM3. 8-bit Micro Controller for Spartan-3, Virtex-II and Virtex-II Pro*. Xilinx Ltd., 2003, October 2003.

[15] S. Reinhardt and S. Mukherjee, "Transient fault detection via simultaneous multithreading," in *27th Int. Symp. on Computer Architecture*, 2000, Proc. Paper, pp. 25–36, Vancuver, Canada, Jun 12-14, 2000.

[16] G. A. Reis, J. Chang, and D. I. August, "Automatic instruction-level software-only recovery," *IEEE Micro*, vol. 27, no. 1, pp. 36–47, 2007.