SEU Evaluation of Hardened-by-Replication Software in RISC-V Soft Processor

Corrado De Sio, Sarah Azimi, Andrea Portaluri, Luca Sterpone
Dipartimento di Automatica e Informatica (DAUIN)
Politecnico di Torino
Turin, Italy
{corrado.desio, sarah.azimi, andrea.portaluri luca.sterpone}@polito.it

Abstract—**The interest of the space industry around soft processors is increasing. However, the advantages in terms of costs and customizability provided by soft processors are countered by the reliability issues deriving by Single Event Effects, especially Single Event Upsets. Several techniques have been proposed to tackle these issues, both at the hardware- and software levels. Software approaches rely on replicating data and computations to cope with SEUs affecting the memory where the binary code is stored. Thanks to open licenses, RISC-V solutions are steadily growing in popularity among the set of available soft processors. In this works, we present a reliability evaluation of four different benchmarks running on the RI5CY soft processor implemented on SRAM-based FPGAs. The reliability of the baseline and hardened-by-replication versions of the software benchmarks are evaluated against SEUs-induced faults both at the software and hardware architecture levels through fault injection campaigns in the microprocessor memory and configuration memory, respectively. Results assess how the adoption of the hardening-by-replication technique at the software level slightly improves reliability against software-related faults but degrades reliability against architectural faults, making it an inefficient solution when it is not combined with hardware robustness.**

Keywords—**Fault injection, Reliability, Reconfigurable, SoC, RISCV, SEU, SRAM-based FPGA.**

I. Introduction

In the last years, programmable-hardware devices, in particular Field Programmable Gate Arrays (FPGAs), have been adopted in many mission-critical applications. Their high performance, along with the advantages they offer in terms of flexibility and costs compared to Application-Specific Integrated Circuits (ASICs), made programmable-hardware devices a suitable choice for automotive and space applications [1][2][3]. A soft microprocessor is one of the cores commonly implemented using programmable hardware. A microprocessor as an IP Core provides an easy way for combining a microprocessor with hardware acceleration, coupling the high performance of hardware with the flexibility granted by the software. Among the available solutions, soft microprocessors based on the RISC-V ISA are attracting a lot of interest in recent years. The open license along with the wide support of the community have made RISC-V solutions (e.g., NOEL-V or Taiga) attractive for space, automotive and avionic industries too [4][5][6]. However, when using a soft microprocessor in mission-critical applications, the reliability issues deriving from the exposure of the devices to ionizing radiation, such as Single Event Upsets (SEUs) should be taken into account.

Several approaches have been proposed for improving application reliability against SEU-induced errors. Even if approaches based on hardware redundancy given proof to be very effective, they are also very costly in terms of design time, area overhead, and power consumption. On other hand, software approaches are easier to apply and less demanding. Software approaches replicate the data in memory and the operations are performed many times using redundant code and verifying the consistency during the execution [7][8]. Even if they are less performing, they usually provide a low-cost reliability improvement against SEUs.

Differently from hardwired microprocessors, soft microprocessors implemented on SRAM-based FPGAs have another criticality due to the presence of the configuration memory (CRAM), that defines the netlist implemented in the programmable hardware. This memory can be corrupted by SEUs in a similar way to the main memory, leading to hardware architectural fault. This paper is dedicated to analyzing the benefits and drawbacks of hardened-by-replication software applications running on soft microprocessors.

The main contribution of this work is evaluating the impact of SEUs in the main and configuration memory of a soft RISC-V soft processor implemented on an FPGA device. In the paper, we evaluate the SEUs occurring in main and configuration memory affecting the baseline and hardened-by-replication versions of a software benchmark suite. Reliability evaluations are based on fault injection campaigns, and the proposed methodology is elaborated in detail in the paper. The twofold analysis allows to comprehensively evaluate the effects of using hardened-by-replication software. Results report that hardening-by-replication techniques at the software level improve reliability against in-memory SEUs only marginally, but they degrade the reliability against hardware architectural faults.

The paper is organized as follows. Section II is dedicated to the background on SEUs in microprocessor memory and configuration memory, and software mitigation techniques. Section III reports related works while Section IV describes the fault injection environment and methodology. In Section V, the experimental analyses are reported along with the obtained results. Finally, Section VI elaborates on conclusions and future works.

II. Background

A. Single Event Upsets in Memories

An SEU is the modification of the content of a memory cell (i.e., a memory bit) caused by the energy released by a particle. This phenomenon corrupts the information stored in the memory cell, producing a fault that can produce errors. Due to the high density of transistor nodes, memories are very

978-1-6654-1609-2/21/$31.00 ©2021 IEEE

sensitive to SEUs. Both hard and soft microprocessors use memory for storing data and machine code for the software execution. Hence, SEUs can corrupt the binary machine code stored in the memory leading to soft errors or misbehaviors in the software application. SEUs can be observed also in the hardware resources of the processing unit, such as flip-flops and registers, even as an effect of transient faults [9]. However, the lower density of memory elements makes this a less common scenario. Soft microprocessors present an additional issue related to their implementation on programmable hardware. To elaborate more, soft microprocessors are implemented using programmable hardware. Programmable hardware consists of a set of basic elements, such as look-up tables, flip-flops, block memories, DSP, and others, that can be configured to implement specific behaviors (e.g., the truth-table for LUTs). Interconnection between basic elements is also programmable and based on a combination of hardwired lines and Programmable-Interconnection-Points (PIPs). Since programmable hardware devices usually rely on an SRAM-based configuration memory for storing configuration data, SEUs affecting configuration memory can produce faults in the programmed hardware modifying the netlist of the implemented cores, soft microprocessor included, producing faults in the hardware architecture, such as open nets, antennas or gates misbehaviors [10][11]. While occasional SEUs and errors are acceptable for some applications, they are not for mission-critical applications where a failure can produce catastrophic outcomes in terms of costs or human lives.

B. *Hardening-By-Replication Software Techniques*

Nowadays, many hardware and software techniques have been explored for satisfying high reliability requirements. Software hardening-by-replication techniques were proposed as one of the first solutions to improve the fault tolerance of critical systems [7][12][13]. Even if software-based fault mitigation has proven to be less effective than demanding hardware techniques like Triple Modular Redundancy (TMR), the simplicity of implementation has resulted in them being commonly implemented. Already in the early 2000s, NASA was recommending the use of software techniques based on replication for improving fault tolerance of mission-critical applications [7]. Software hardening-by-replication techniques are based on the redundancy of data and computations. To cope with SEUs affecting data in memory, input data is replicated in the memory. The hardened-by-replication software is written such as it executes a computation multiple times using different in-memory copies of the same input and intermediary results. Depending on the required granularity and tolerable overhead, a number of *detection and correction* checkpoints are inserted in the code. In the *detection and correction* checkpoints, the values of the temporary results are compared to each other to detect errors in the computations. Erroneous values are then corrected by majority voting during each detection and correction checkpoint as well as on the final output. A conceptual schema is illustrated in Fig. 1. Computational data flow illustrated in (a) is triplicated in (b) and checkpoints have been added to correct errors affecting one of the data paths through comparison.

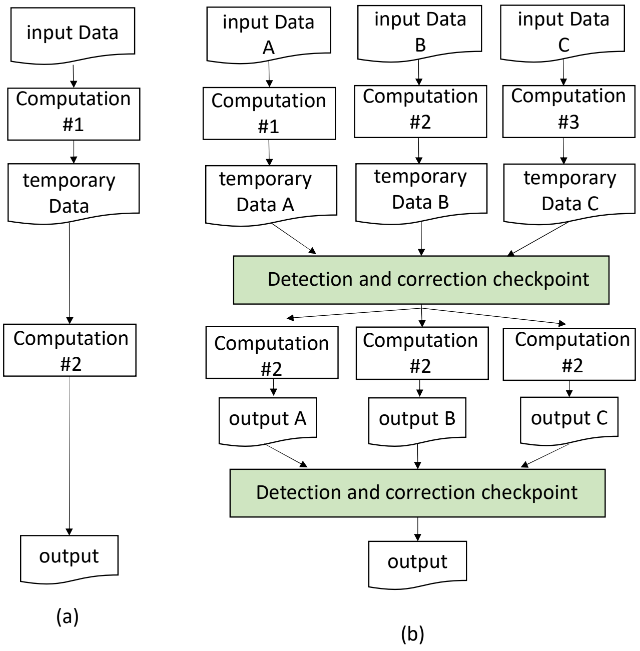

Fig. 1. Schema of the code (a) Unhardened (b) Hardened-by-Replication of data and operation and with insertion of detection and correction checkpoint.

III. Related Works

Several works addressed the reliability issues of software applications and hardware platforms. About the effectiveness of software mitigation techniques, the authors at [8] proposed a methodology for detecting soft errors in code and data exploiting a software replication approach. In [14], SWIFT is proposed as a performing approach for software fault detection relying on unused instruction-level parallelism resources. A software technique implementing both detection and correction based on data and code replication has been presented in [15]. In [16], both hardware and software techniques are evaluated by fault injection campaigns against SEUs affecting microprocessors. Even if FPGAs are involved in the analysis, the authors evaluated faults affecting the storage elements without considering configuration memory. On the soft microprocessor side, the authors at [17] presented a comprehensive analysis of the SEU-induced errors on a set of software benchmarks running on RISCV soft microprocessors. A similar analysis has been published in [18]. However, to the best of our knowledge, no work evaluated the effect of hardening-by-replication software techniques against SEUs affecting configuration memory of soft microprocessors implemented on SRAM-based FPGAs.

IV. Analysis Environment and Methodology

The current section elaborates on the hardware platform, soft microprocessor, software benchmark applications, and fault models involved in the experimental analyses. Additionally, it illustrates the fault injection platform adopted and extended for performing the reliability analyses as well as the fault injection methodology. The RISC-V soft microprocessor implemented on a hardware configurable device is the evaluated hardware platform. A set of software applications are adopted as the benchmark suite for the reliability evaluations. Each software application has been implemented in both unhardened, aka baseline, and hardened-by-replication versions. Using the developed fault injection platforms, the fault model is emulated in the memory of the microprocessor and the configuration memory of the hardware platform while the software of the benchmark suite is executing.

A. RISC-V Soft Microprocessor

The RISC-V is an open-source standard Instruction Set Architecture (ISA), supported by RISC-V Foundation. RISC-V-based soft microprocessors are an attractive solution that can keep costs down by combining open licensing with the use of open-source cores to be implemented on programmable devices, without the costs need for a semiconductor fabrication plant. In addition, the open architecture has made these solutions easier to customize (in terms of cost and difficulty) than traditional solutions such as licensed and hard microprocessors. Among open-source solutions, we selected PULPissimo as the microcontroller architecture for reliability evaluation analysis [19]. PULPissimo is a single-core platform including the RI5CY Core, developed by the PULP project, a collaboration between the ETH Zurich and the University of Bologna. PULPissimo is a microcontroller implementable on FPGA devices, and it is designed for high energy efficiency. RI5CY core is an in-order single-issue core. It is provided with 4 pipeline stages and supports RV32I, RV32C, RV32M, and RV32F instruction set. For the purpose of this work, we implemented PULPissimo on a Nexys Video Artix-7 platform. Table I reports the device utilization when implementing the PULPissimo platform.

TABLE I. RESOURCES UTILIZATION OF PULPISSIMO

Resources	Available [#]	Used [#]	Utilization [%]
Logic Slices	33,650	14,150	42.05
Flip-Flops	269,200	21,531	8.00
Memories	365	128	35.07
DSPs	740	12	1.62

B. Software Benchmark Application

As software applications, a set of four software benchmarks have been adopted. Applications have been selected to cover different domains, such as signal and image processing. In this paper, they are referred to as:

- *CoreMark*: CoreMark software implements the CoreMark benchmark of EMBC. It involves list processing, matrix manipulation, state machine execution, and cyclic redundancy check.
- *Dhrystone*: Dhrystone is a performance benchmark. It focuses on string processing, without the use of any floating-point operation.
- *FFT*: FFT software implements the Fast Fourier Transform. widely used in signal processing. The specific implementation has been selected from MiBench Benchmark Suite.
- *Sobel*: Sobel software implements the Sobel operator, used in image processing for edge detection.

In order to evaluate the benefits of the software hardening-by-replication approach, a hardened version of each software application has been developed. In particular, according to [7], single-version software fault tolerance techniques have been applied to each software application. Input data, variables, and functions have been triplicated in the memory. The software has been modified to perform the same operations sequentially on the different data copies stored in the memory. *Detection and correction* checkpoints have been inserted during statement execution.

C. Fault Models

SEUs are one of the main sources of errors, especially when memories are involved. Due to its architecture and technology, soft microprocessors have a traditional microprocessor memory (e.g., main memory and cache levels) and configuration memory. These memories can be both affected by SEUs, causing very different faults and eventually errors. In this paper, we evaluated the reliability of applications running on soft processors against SEUs in microprocessor memory and configuration memory, separately. An SEU is a bit flip in the content of a memory cell. An SEU in the microprocessor memory may corrupt either data or code segments of the program loaded in that part of memory. This can lead to different outcomes, such as errors (e.g., data value corruption) or system halt. On other hand, SEUs in configuration memory will introduce faults directly in the hardware architecture of the soft processors. For example, if an SEU in configuration memory introduced a fault in the ALU, arithmetic operations performed by software applications could be affected by errors.

D. Fault Injection Platform and Methodology

Two different fault injection platforms have been adopted for emulating SEUs in microprocessor memory and configuration memory.

SEU in the main memory has been emulated acting directly on the Executable and Linking Format (ELF) to be loaded in the main memory of the microprocessors. In detail, a python-based fault injection platform has been developed for performing the fault injection process, loading the binary code in the memory of the PULPissimo microcontroller, and collecting the output of the injected applications. The fault injection step is performed by flipping a bit of the ELF so that a faulty ELF file is generated. Using Open On-Chip Debugger (OpenOCD) and GDB, the faulty ELF file is loaded in the memory of the microcontroller by communicating with the RISC-V debug module via the JTAG interface. All these steps are performed automatically by the platform, which automatically instruments the OpenOCD and GDB tools. The platform will wait for the results of the software computation on the serial port. A timeout mechanism is used to handle the halt of the processor due to injected faults.

For emulating SEUs in the configuration memory, the PyXEL platform has been used [20]. PyXEL is a python-based platform for performing FPGA fault injection campaigns, able to manipulate FPGAs bitstreams to be loaded in configuration memory to inject faults. In order to support the reliability analysis workflow presented in this work, PyXEL has been extended for supporting Artix-7 XC7A200T FPGA. It has been used to emulate SEUs in the configuration memory by corrupting a bit of the bitstream to be loaded in the configuration memory. Additionally, PyXEL automatizes the steps for configuring the FPGA platform with the bitstream implementing the RISC-V soft microprocessor and platform resetting in case of a halt due to fault injection, as well as the steps for loading and running the software applications on the soft microprocessors and collecting the results.

V. Experimental Analyses and Results

We carried out reliability analyses for evaluating the benefits and drawbacks of applying hardening-by-replication software techniques to software applications running on a soft microprocessor. The baseline and hardened software benchmark applications have been evaluated against SEUs in the soft microprocessor memory and the hardware-configurable platform configuration memory through fault injection campaigns. The fault injection platforms reported in Section IV have been used in the reliability evaluation for the two fault models. Results have been collected, categorized, and discussed. Reliability analysis involved a RI5CY soft microprocessor implemented within the PULPissimo microcontroller on the Artix-7 XC7A200T FPGA. The software benchmarks run as bare-metal, without any operative system.

A. Cross Sections

In order to perform an accurate radiation analysis, we performed the radiation characterization of the memory cell, representing the used technology in the memory cell where the ELF binary file is loaded as well as the configuration memory. The characterization is performed in terms of cross-section, defined as the radiation sensitivity of the cell with respect to the physical characteristic of the technology. We developed the electrical model of the memory cell, exploiting the FreePDK physical library tuned for 28 nm, as the technology of the used hardware adopting the electrical Predictive Technology Model (PTM) for bulk CMOS. Using the K-layout tool, the layout description of the memory cell has been designed and extracted in terms of Graphic Data System-II (GDS-II). Based on the netlist and layout of the memory cell, we have performed a radiation analysis using our in-house Monte Carlo bases simulation tool, described in detail in [21], using the Heavy Ion Profile related to the Université Catholique de Louvain (UCL) facility [22]. We have performed a simulation of 10,000 particles. The obtained cross-section is shown in Fig. 2.

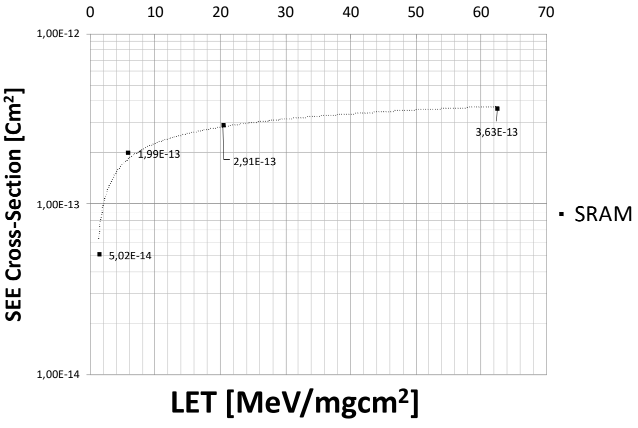

Fig. 2 Single Event Upset (SEU) cross-section [cm2] for static radiation analysis of memory cell in 28 nm.

B. Errors Categorization

As a result of the fault injections, different misbehaviors may occur. Errors are detected through a comparison of the outcomes of the fault injection experiments and the golden runs (i.e. where each faulty-free software application has been executed on the faulty-free soft microprocessor). The collected results have been categorized into three categories: *correct, silent data corruption,* and *halt.* They are defined as follow:

- *Correct*: The task terminates correctly and the output matches the golden one.
- *Silent Data Corruption*: The task terminates but the output does not match with the golden one.
- *Halt*: the soft microprocessor does not complete the task. It can be due to different causes, such as infinite loops, illegal code instructions, or others. It can be generated either by a fault in the binary code or at the hardware architecture level.

Error Rate is defined as the percentage of results that deviate from the nominal behavior, in other words, the percentage of the outcomes that are categorized as Silent Data Corruption (SDC) or Halt.

C. Baseline Software Evaluation against SEU in Memory

The reliability of the baseline software against SEUs in the memory has been evaluated through a fault injection campaign. We carried out 10,000 experiments for each of the software benchmarks reported in Section IV. SEU coordinates (i.e., the bit to flip in the binary code) have been chosen independently and randomly for each experiment. The errors generated by the injected faults have been detected by comparison between the outputs of the faulty and faulty-free binary code. Results have been classified into three categories. The results are reported in Fig. 3, while the categories are illustrated in Table II.

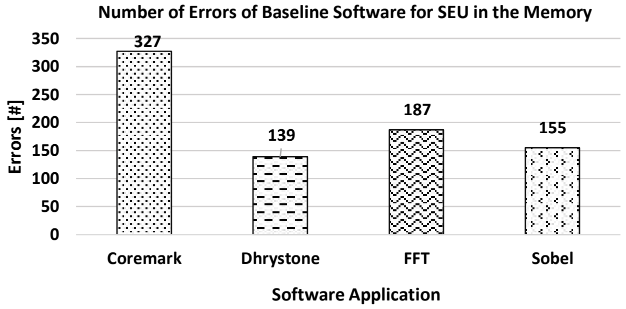

Fig. 3. Number of Errors of Baseline Software out of 10,000 SEUs in the Microprocessor Memory.

Table II. Baseline Software against SEU in Memory

Software	Correct [#]	SDC [#]	Halt [#]
Coremark	9,673	172	155
Dhrystone	9,861	69	70
FFT	9,813	80	107
Sobel	9,845	81	74

D. Baseline Software Evaluation against SEU in CRAM

After the evaluation of baseline software against SEUs affecting the microprocessor memory, we analyzed also the effects of an SEU in the configuration memory of the FPGA implementing the soft microprocessor. The corruption of the configuration memory content introduces errors in the architecture of the netlist implemented on the programmable hardware. Due to the characteristics of programmable hardware, where only a subset of the resources are used and programmed (as reported in Table I), the error rate resulting from SEUs in configuration memory is usually low, since only a subset of the bits of the configuration memory is usually used by the design. However, since these errors permanently affect the microprocessor operation until the next reconfiguration or power cycle, they play a critical role in the reliability evaluation. The current fault injection campaign consists of 10,000 faults, injected singularly and randomly in the configuration memory. Each software benchmark has been evaluated while running on each of the faulty configurations. Since different software uses different logic of the soft microprocessor, they will be characterized by different error rates even if running on the same faulty configurations. Similar to the previous campaign, results have been collected and categorized. Fig. 4 summarizes the application failures, while Table III reports results classification**.**

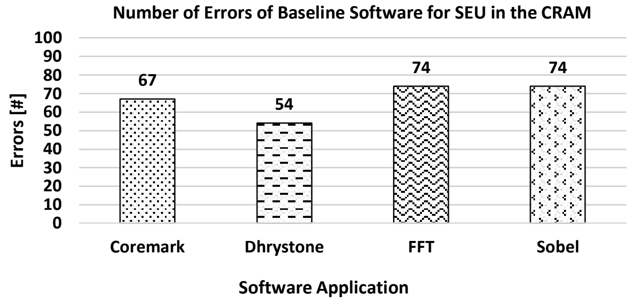

Fig. 4. Number of Errors of Baseline Software out of 10,000 SEUs in the Configuration Memory.

TABLE III. BASELINE SOFTWARE AGAINST SEU IN CRAM

Software	Correct [#]	SDC [#]	Halt [#]
Coremark	9,933	20	47
Dhrystone	9,946	12	42
FFT	9,926	28	46
Sobel	9,926	10	64

E. Hardened Software Evaluation against SEU in Memory

The hardened version of the software benchmarks has been used in similar fault injection campaigns for evaluating the effects introduced by the hardening technique. The first analysis on hardened software resembles the one reported in section V-B. We performed 10,000 experiments on each hardened software application. We evaluated the effect of SEUs in memory by flipping a bit in the binary code in each experiment and evaluating the results. The coordinates where to inject the fault have been chosen randomly for each experiment. Fig. 5 illustrates the obtained results. Results have been categorized accordingly with their behavior in Table IV. For the experiments carried out in Section V-B, the overall error rate slightly decreases for all the applications. From the categorization represented in Table IV, we observe that while SDC errors decreased, Halt errors slightly increased. This is reasonable since, without an operating system, exceptions (e.g., OPCODE exceptions) caused by fault injection cannot be handled.

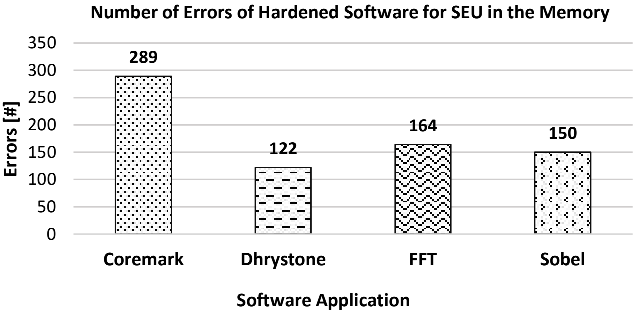

Fig. 5. Number of Errors of Hardened Software out of 10,000 SEUs in the microprocessor.

TABLE IV. HARDENED SOFTWARE AGAINST SEU IN MEMORY

Software	Correct [#]	SDC [#]	Halt [#]
Coremark	9,711	122	167
Dhrystone	9,878	41	81
FFT	9,836	56	108
Sobel	9,850	80	70

F. Evaluation of Hardened Software against SEU in CRAM

The last fault injection campaign is for evaluating the hardened-by-replication software against SEUs affecting the configuration memory. The hardened-by-replication software applications have been evaluated against the same faults injected in Section V-C. Results are illustrated in Fig. 5. Results categorization is reported in Table V.

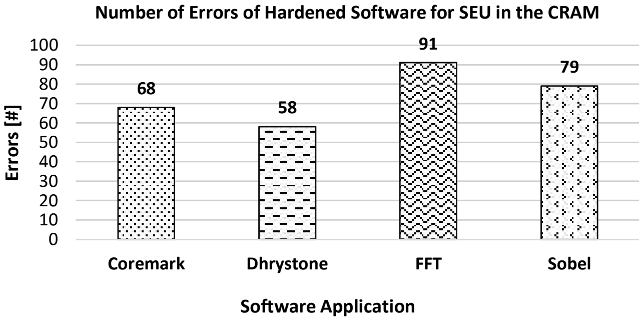

Fig.6. Number of Errors of Hardened Software out of 10,000 SEUs in the Configuration Memory.

TABLE V. HARDENED SOFTWARE AGAINST SEU IN CRAM

Software	Correct [#]	SDC [#]	Halt [#]
Coremark	9,932	19	49
Dhrystone	9,942	11	47
FFT	9,909	30	61
Sobel	9,921	9	70

G. Results Analysis

As a result of the comparison between the reliability of the baseline and hardened software, some interesting results should be noticed. Firstly, results report software hardening slightly increases the reliability of all the applications against SEUs affecting processor memory. However, the trend is not the same with SEUs in configuration memory, which reports reliability degradation. Reasonably, there are two causes for the observed behavior. Firstly, software replication techniques are not useful when errors affect hardware elements of the microprocessors. In particular, performing the same operation twice on the same faulty hardware will likely produce the same erroneous output. There are some exceptions to this (e.g., errors generated in the reading of a specific memory cell can be corrected by reading replicated data that are stored in different memory cells). However, due to this effect, the mitigation benefits are reduced compared to faults affecting microprocessor memory. Secondly, the introduction of the code for implementing the *detection and correction checkpoint* could stimulate sections of the

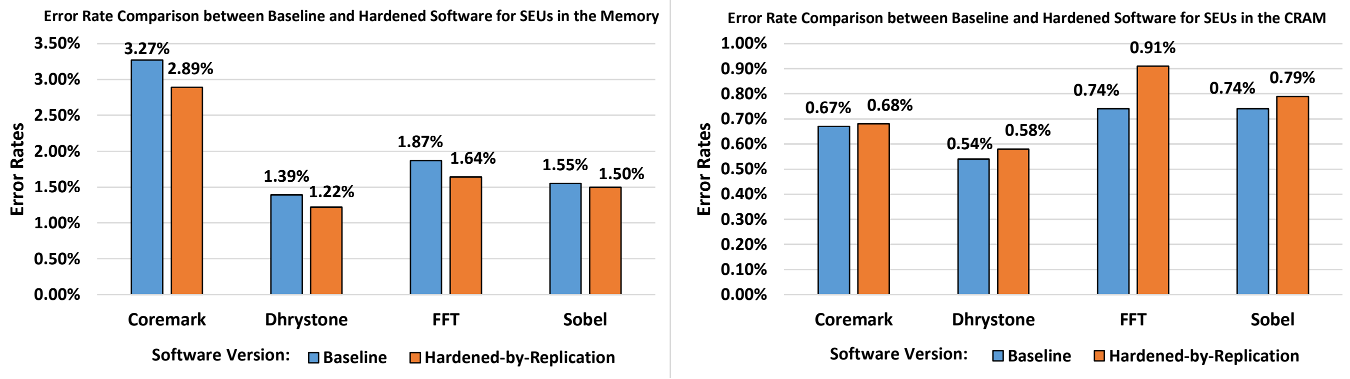

Fig. 5. Comparison between Error Rates of Baseline and Hardened Software for SEU in the processor memory and in the configuration memory.

electronic circuit that were not used by an unhardened version of the software. This could cause that hardware faults that formerly were not propagated to the application outputs, now produce a deviation from the nominal behavior of the software application.

VI. Conclusions and Future Works

In the current work, we initially provided an evaluation of the reliability of software applications running on a soft microprocessor against SEU affecting configuration memory and microprocessor memory. Then, the benefits of a hardening-by-replication software technique have been evaluated comparing the reliability of the hardened software with the baseline implementation. Results highlighted some interesting behavior. The Hardening-by-replication software technique produces an increase of the reliability against SEU at the software level (i.e., SEU in microprocessor memory). However, it does not provide any improvement against hardware-level faults caused by SEU in the configuration memory of the soft microprocessor. In fact, the hardened version of the software showed a decrease in reliability in all the evaluated applications against SEUs in the configuration memory. In the future, we plan to evaluate the effects of the software hardening technique during radiation test experiments. Additionally, we want to extend reliability evaluation analysis of baseline and hardened-by-replication software running on a hardware-hardened platform.

References

[1] S. Azimi, *et al.,* “A new CAD tool for Single Event Transient Analysis and mitigation on Flash-based FPGAs”, *Integration*, Volume 67, 2019, pp. 73-81, ISSN 0167-9260, DOI: 10.1016/j.vlsi.2019.02.001.

[2] A. Hofmann, R. Wansch, R. Glein and B. Kollmannthaler, "An FPGA based on-board processor platform for space application," *2012 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, 2012, pp. 17-22, doi: 10.1109/AHS.2012.6268653.

[3] L. Chen *et al.*, "Surrounding Vehicle Detection Using an FPGA Panoramic Camera and Deep CNNs," in *IEEE Transactions on Intelligent Transportation Systems*, vol. 21, no. 12, pp. 5110-5122, Dec. 2020, DOI: 10.1109/TITS.2019.2949005.

[4] D. A. Santos, *et al.*, "A Low-Cost Fault-Tolerant RISC-V Processor for Space Systems," *2020 15th Design & Technology of Integrated Systems in Nanoscale Era (DTIS)*, 2020, pp. 1-5, DOI: 10.1109/DTIS48698.2020.9081185.

[5] A. Waterman, *et al.*, "The RISC-V instruction set manual volume i: Base user-level isa", *EECS Dept.*, vol. 116, 2011.

[6] J. Andersson, "Development of a NOEL-V RISC-V SoC Targeting Space Applications," *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*, 2020, pp. 66-67, DOI: 10.1109/DSN-W50199.2020.00020.

[7] Torres-pomales, Wilfredo. (2000). Software Fault Tolerance: A Tutorial.

[8] M. Rebaudengo, M. Sonza Reorda, M. Torchiano, and M. Violante, "Soft-error detection through software fault-tolerance techniques," *IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (EFT'99)*, 1999, pp. 210-218.

[9] S. Azimi *et al.*, “On the analysis of radiation-induced Single Event Transient on SRAM-based FPGAs”, *Microelectronics Reliability*, vol. 88-90, 2018, pp. 936-940, ISSN 0026-2714, DOI: https://doi.org/10.1016/j.microrel.2018.07.135.

[10] B. Du *et al.*, "Ultrahigh Energy Heavy Ion Test Beam on Xilinx Kintex-7 SRAM-Based FPGA," in *IEEE Transactions on Nuclear Science*, vol. 66, no. 7, pp. 1813-1819, July 2019, DOI: 10.1109/TNS.2019.2915207.

[11] L. Sterpone *et al.*, "A Novel Error Rate Estimation Approach for UltraScale+ SRAM-based FPGAs," *2018 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, 2018, pp. 120-126, DOI: 10.1109/AHS.2018.8541474.

[12] L. Sterpone *et al.*, "A selective mapper for the mitigation of SETs on rad-hard RTG4 flash-based FPGAs," *2016 16th European Conference on Radiation and Its Effects on Components and Systems (RADECS)*, 2016, pp. 1-4, doi: 10.1109/RADECS.2016.8093152.

[13] S. Azimi and L. Sterpone, "Digital Design Techniques for Dependable High Performance Computing," *2020 IEEE International Test Conference (ITC),* 2020, pp. 1-10, doi: 10.1109/ITC44778.2020.9325281.

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

[15] M. Rebaudengo, M. S. Reorda, and M. Violante, "A new software-based technique for low-cost fault-tolerant application," *Annual Reliability and Maintainability Symposium,* 2003, pp. 25-28.

[16] C. Bolchini, *et al.* "Software and Hardware Techniques for SEU Detection in IP Processors" *Journal of Electronic Testing, 24*, **35**–44,2008.

[17] L. A. Aranda, *et al.* "Analysis of the Critical Bits of a RISC-V Processor Implemented in an SRAM-Based FPGA for Space Applications" *Electronics* 9, no. 1: 175, 2020.

[18] A. Ramos, J. A. Maestro, P. Reviriego “Characterizing a RISC-V SRAM-based FPGA implementation against Single Event Upsets using fault injection”, *Microelectronics Reliability*, Volume 78, 2017, Pages 205-211.

[19] P. D. Schiavone, *et al.*, "Quentin: an Ultra-Low-Power PULPissimo SoC in 22nm FDX," *2018 IEEE SOI-3D-Subthreshold Microelectronics Technology Unified Conference (S3S)*, 2018, pp. 1-3, DOI: 10.1109/S3S.2018.8640145.

[20] L. Bozzoli, *et al.*, "PyXEL: An Integrated Environment for the Analysis of Fault Effects in SRAM-Based FPGA Routing," 2018 International Symposium on Rapid System Prototyping (RSP), 2018, pp. 70-75, DOI: 10.1109/RSP.2018.8632000.

[21] L. Sterpone, *et al.*, "A 3-D Simulation-Based Approach to Analyze Heavy Ions-Induced SET on Digital Circuits," in IEEE Transactions on Nuclear Science, vol. 67, no. 9, pp. 2034-2041, Sept. 2020.

[22] A. O. Akhmetov *et al.*, "IC SEE Comparative Studies at UCL and JINR Heavy Ion Accelerators," 2016 IEEE Radiation Effects Data Workshop, Portland, OR, USA, 2016, pp. 1-4.