A Low-Cost Fault-Tolerant RISC-V Processor for Space Systems

Douglas Almeida Santos*†, Lucas Matana Luza†, Cesar Albenes Zeferino*,
Luigi Dilillo†, and Douglas Rossi Melo*†
* Laboratory of Embedded and Distributed Systems (LEDS), University of Vale do Itajaí, Brazil
† Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier (LIRMM), France
douglasas@edu.univali.br, lucas.matana-luza@lirmm.fr, zeferino@univali.br, dilillo@lirmm.fr, drm@ieee.org

***Abstract*—Embedded processors have been used in a diversity of applications, such as consumer electronics, home appliances, and automation systems. Another area that embedded processors can be adopted is space systems, which demand fault-tolerant components to deal with the environmental hostility. This work presents a low-cost fault-tolerant implementation of the RISC-V architecture, an emerging open industry standard for the building of embedded processors. The proposed implementation employs physical and information redundancy to reduce error propagation, with competitive silicon and power overheads when comparing with other RISC-V implementations.**

***Index Terms*—Spatial Systems, RISC-V, Digital Systems, Fault Tolerance**

I. Introduction

Space systems operate in a harsh environment and are exposed to space radiation and extreme temperatures, as well as vacuum and lack of gravity. This hostility of the space environment can lead to transient, permanent, or intermittent faults that can affect the functioning of computational systems [1]. Single and multiple bit upsets, per example, may lead to catastrophic failures in critical systems. Therefore, these systems must be designed to deal with the characteristics of the space environment using fault tolerance techniques.

In order to improve the reliability of space systems, several techniques can be used to protect systems against radiation effects. The utilization of redundancies [2] is crucial to reliable systems and has three classification types. Spatial redundancy is considered the replication of the same module seeking to compare the results. Temporal redundancy is the reuse of the same module repeatedly in order to compare the results and get the right result. Information redundancy is the addition of redundant bits to data to detect and even correct when it has been affected by an error.

There are several processors designed for use in the space environment, notably synthesizable soft-cores. The designs of these processors apply fault tolerance techniques to protect their circuits. LEON4-FT [3] employs error-correcting code (ECC) to protect its RAM blocks against SEU (single-event upset) transient faults. CFTP (configurable fault-tolerant processor) [4] uses triple-modular redundancy (TMR) in the entire processor core. MIPS Crypto [5] applies TMR and matrix encoding techniques to protect its memory buffer units. In [6], the authors employed Hamming code to protect registers and TMR for providing fault tolerance to the messaging interface. These examples illustrate how fault tolerance techniques have been employed in SPARC- and MIPS-based processors.

One emerging processor architecture that is becoming an industry standard is RISC-V [7]. Its design is similar to MIPS and is an optimized instruction-set architecture (ISA) that aims at simplifying the processor implementation. Although there are already several RISC-V soft-cores available [8]–[17], there are just a few RISC-V fault tolerant processors: the RISC-V-based SoC (system on chip) developed by [18] contains a pipelined RV32I core and protects the internal core architecture using Hamming ECC at the memory elements and NMR (N-modular redundancy) at combinational elements that cannot be protected by ECC. The work from [19] adapted the Taiga processor [15] to improve its reliability, for that they used the BL-TMR [20] software, which identifies the most critical elements from the netlist and applies TMR with a voter as needed.

This work presents an implementation with the combination of spatial and information redundancies to improve processor reliability. Our RISC-V soft-core implementation employs Hamming code to protect the memory elements and TMR to protect the arithmetic and logic unit (ALU) and the control unit. We are focusing on SEU and SET (single-event transient) single-event effects (SEE) at the processor core, disregarding the instruction and data memories. The dose-effect of radiation is not treated in this work.

The remainder of this paper is organized as follows. Section II presents the primary features of this ISA. Section III describes the proposed fault-tolerant implementation of the RISC-V processor. Section IV describes the materials and methods employed for this work. Section V discusses the results obtained and Section VI presents the final remarks.

II. RISC-V

RISC-V is an open ISA developed by Waterman et al. [7]. Currently, the architecture specification is still under development [21], but its 32-bit integer instruction set (RV32I) is already in its final version.

A. Features

This architecture has three base instruction sets: (i) RV32I, a set of 47 instructions complete enough to satisfy the basic

978-1-7281-5426-8/20/$31.00 ©2020 IEEE

requirements of modern operating systems; (ii) RV32E resembles the previous set, but is designed to use only 15 registers; and (iii) RV64I, similar to the RV32I set, differs only in the width of integer registers and program counter (PC) [22].

RISC-V architecture is designed to simplify processor implementation. Instruction coding is extremely regular because the memory model is straightforward, and it has no complex instructions for accessing data memory.

A feature of RISC-V implementations is the small footprint of the minimal cores, which are much smaller compared to similar ones, such as Advanced RISC Machine (ARM) and x86. However, the difference is not significant in larger capacities [23].

B. RISC-V distributions

There are several RISC-V processors available for use. Examples include Ibex [8] and PicoRV32 [9] processors, which are implementations focused on low utilization of logical resources, and Ariane [10] processors, Berkeley Out-of-Order Machine (BOOM) [11], and RI5CY [12], which are more complete implementations of RISC-V, targeting devices with higher computing power.

III. Fault-Tolerant RISC-V

Our proposal of processor is based on the RISC-V unprivileged specification [24]. We implemented the RV32I instruction set, except the synchronization instructions and environment calls. We focus our work on using the least amount of resources possible. For this reason, our processor uses a single-cycle organization (or micro-architecture) [22], which allows us to reduce the required registers. Thus, the developed processor has five primary units: (i) instruction fetch; (ii) instruction decode; (iii) execution; (iv) memory access; and (v) write-back.

A. Hardware Design

The instruction fetch unit has in its structure the PC register, a 32-bit adder, and the logic circuits for the conditional and unconditional branches. The adder increments the program counter in case of a sequential execution or adds the offset when a conditional branch is executed.

The control unit integrates the instruction decode unit and is responsible for decoding the instruction and defining the operation to be performed by the datapath. We implemented both the main and ALU control units as a single component, simplifying the subsequent implementation of fault tolerance techniques.

The execution unit performs reading and writing from/into the register file, as well as the arithmetic operations. The arithmetic and logical operations executed by the ALU are: add, shift (left/right logical and right arithmetic), set on less than, AND, OR, and XOR.

The memory access unit performs read and write operations from/into data memory. The RISC-V specification describes that accesses with 8-, 16-, and 32-bit data word widths can be performed, and all readings resulting in 32-bit width data. According to the instruction executed, the data signal can be extended or not.

Writing into the register file is done by the write-back unit. This unit selects the value to be written to the specified register, which may be the result of the ALU, a variable read from the data memory, or an immediate operand of the instruction.

B. Fault Tolerance Application

We have implemented the fault tolerance techniques at the organization level, allowing the programmer to use the same ISA without worrying about how the processor is implemented. Our implementation focused only on SEE in the processor organization and did not consider clock-tree and RAM issues.

The ALU and the control unit were protected using TMR. Thus, these blocks have been triplicated, and each of their outputs goes through a bitwise simple majority voting circuit. Fig. 1 presents a diagram block with three instances of the ALU having their outputs analyzed through the voter. The voter is susceptible to faults since it is not hardened.

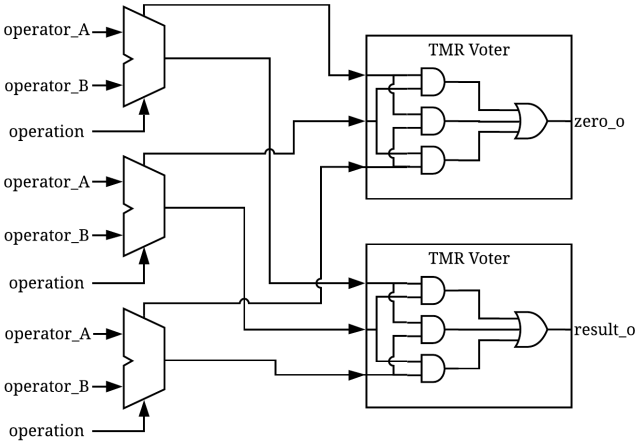

Fig. 1. TMR architecture in ALU.

For Hamming implementation, we increased the width of all registers by six bits and added the encoder and decoder. The encoder block uses XOR operators to calculate the parity bits of the data to be written into the register and concatenates them to the encoded data. The decoder block is responsible for verifying the parity of the encoded data, and when it detects an error, it fixes it by inverting its value.

Figs. 2 and 3 present the Hamming implementations in the instruction fetch unit and register file, respectively. In the instruction fetch unit, we protected the PC register with an encoder and a decoder. In the register file, we placed an encoder at the write port and a decoder at each read port.

IV. Materials and Methods

Implementation was performed using VHDL and a platform-independent approach; no vendor-specific Intellectual Property (IP) blocks were used. This approach allows a designer to reuse the processor on devices from different

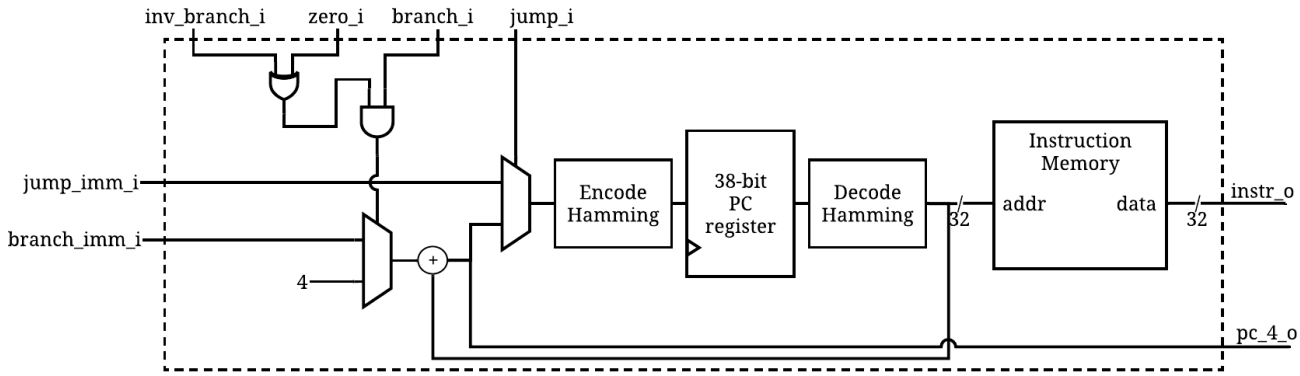

Fig. 2. Instruction Fetch unit using Hamming.

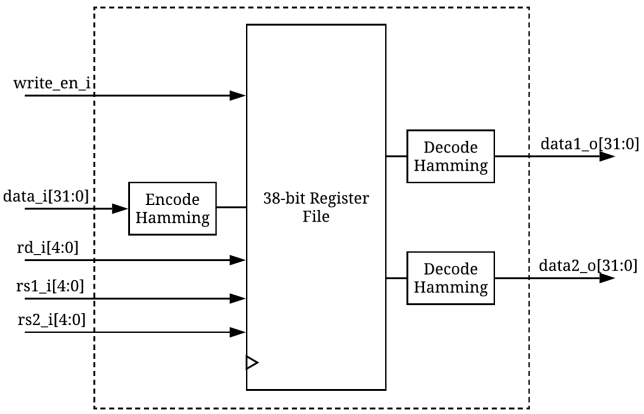

Fig. 3. Register file using Hamming.

manufacturers. We first implemented a non-hardened RISC-V soft-core to be used as a baseline to evaluate the resource overhead and as a golden model to evaluate the improved resilience obtained through the fault tolerance techniques.

To collect synthesis data, we used the Xilinx Vivado Design Suite 2019.1 and the Zynq ZC7020 SoC device. The metrics used include the number of LUTs (Look-up Tables) and flip-flops (FFs), the maximum operating frequency (F_{max}), and the dynamic power dissipation (P_{dyn}) with all the processors running at 50 MHz.

We used the Mentor Graphics ModelSim simulator to verify the developed processor and evaluate its resiliency. To simulate a fault injection, we employed the scripts proposed in [25]. The codes used in the execution of all algorithms were compiled with a GNU Compiler Collection (GCC) adapted for the RISC-V processor [26].

The efficiency of the fault tolerance techniques implemented was assessed by injecting SEU and SET faults. We ran each experiment 100 times, and, in each of them, we injected a predefined number of faults (1 or 10) into the non-hardened and hardened versions. The non-hardened version consists of the processor without Hamming and TMR, while the hardened version applies Hamming to the PC register and the register file and TMR in the ALU and control unit.

The script was run on the different configurations, applying fault tolerance to component combinations, and then verifying the number of errors propagated in each configuration. This approach allowed obtaining an analysis of the most critical components for processor operation.

The benchmark algorithms, which were used for verification and testing of the processor, are the following ones: (i) vector addition, an algorithm that adds two 300-element one-dimension arrays and stores the result in a third array; (ii) CCSDS-123, a hyperspectral image compression algorithm based on the Consultative Committee for Space Data Systems (CCSDS) standard and similar to that implemented by [27]; (iii) the Coremark benchmark, which tests several processor components and is considered efficient for fault tolerance testing [28].

SET events affect combinational logic. Therefore, in order to simulate an error caused by a SET event, the external and internal signals from the entire processor core were filtered out. The injection of this fault consists in inverting and freezing the signal of a random bit at a random moment.

SEU events affects data stored in registers. So each fault injection for this effect was made by inverting one bit of the data stored in a random register between PC and the 31 registers of the register file.

V. Experimental Results

A. Synthesis results

Using fault tolerance techniques increases the number of logical resources used by the processor. Table I compares four different configurations. It is worth noting that in the configuration applying only TMR to the ALU and control unit, the number of LUTs is almost 1.5 times higher than in the non-hardened processor. At the same time, the use of FFs remains the same, given these circuits are purely combinational. In the configuration in which only the Hamming technique is applied, the number of FFs increases due to the addition of six parity bits to each one of the 32 processor registers, and the extra LUTs are due to the encoding and decoding blocks.

TABLE I
SYNTHESIS RESULTS

Configuration	LUTs	FFs	F_{max} (MHz)	P_{dyn} (mW)
Non-hardened	1 613	1 024	74.65	146
TMR	2 387	1 024	66.50	151
Hamming	1 854	1 216	54.77	162
TMR and Hamming	2 748	1 216	49.99	171

The maximum operating frequency decreases as additional circuits increase the critical path. In the non-hardened configuration, the processor can operate at 74.65 MHz. When applying TMR, there is a drop of approximately 10% in the maximum frequency. On the other hand, when applying only the Hamming technique, the degradation is 26% when compared to the non-hardened processor. Finally, by combining the two techniques, the maximum operating frequency is about 50 MHz, 33% lower than the non-hardened configuration.

Regarding power dissipation, there is a gradual increase ranging from 3% when applying TMR to 17% when combining the two techniques. It is worth noting that all processors are operating at 50 MHz, which is the maximum operating frequency of the slowest configuration.

Table II presents costs in terms of resources related to ours and other low-cost non-hardened RISC-V soft-cores. For a fairer comparison with these processors, their control and status registers (CSR) were disregarded as our implementation does not address these registers.

TABLE II
COMPARISON WITH LOW-COST RISC-V PROCESSORS

Soft-core	LUTs	FFs	P_{dyn} (mW)
Ibex [8]	1 680	1 339	135
PicoRV32 [9]	1 580	1 432	119
mRISCV [16]	2 460	2 014	116
DarkRISCV [17]	2 470	2 262	145
This Work	1 613	1 024	146

Given that the processor proposed in this work was developed with a focus on low resource occupancy, especially for sequential logic, the number of LUTs is lower than most of the other processors, and the use of FFs is lower than all the other RISC-V implementations. Regarding all soft-cores operating at 50 MHz, our implementation is the one that dissipates the most power, however, at a similar level to another processor.

Between the RISC-V fault-tolerant cores, our implementation has the smallest overhead ratio, as shown in Table III. While the processor cores from [18] and [19] got a LUTs utilization overhead of 5.96x and 5.64x, respectively, our implementation has an overhead of 1.7x. As for FFs utilization overhead, [18] achieved an overhead of 3.7x, and [19] achieved an overhead of 3.0x, while our implementation has an FFs overhead of 1.19x. These results were achieved because our implementation is simplified to reduce the number of elements to be hardened, such as pipeline registers, which we do not include.

TABLE III
COMPARISON WITH FAULT-TOLERANT RISC-V PROCESSORS

Processor core	LUTs overhead ratio	FFs overhead ratio	F_{max} (MHz)
FT impl. F.W. Heida [18]	5.96x	3.70x	36.10
TMR Taiga [19]	5.64x	3.00x	227.20
TMR and Hamming	1.70x	1.19x	49.99

B. Fault Tolerance - Results

Concerning the simulated SEU faults, we consider only the effects in the processor while the effects in the FPGA configuration memory are not treated. Table IV presents the error propagation rate when injecting SEU faults, which affect the registers. In the first experiment, one fault was injected at each execution in moments and registers randomly chosen. The vector addition algorithm propagated 64 errors, the CCSDS compressor propagated 71 errors, and Coremark propagated 73 errors. By applying fault tolerance techniques, all algorithms performed without error propagation, as Hamming can correct all single errors. By injecting ten faults at random moments and registers, all algorithms had an error rate close to 100% in the non-hardened processor. However, in the hardened version of the processor, Hamming was able to mask 65% of errors when performing vector addition, 97% of errors when running CCSDS-123, and 94% when executing Coremark.

TABLE IV
ERROR PROPAGATION FOR SEU

Algorithm	1-fault injection		10-fault injection	
	Non-hardened	Hardened	Non-hardened	Hardened
Vector addition	64	0	99	35
CCSDS 123	71	0	98	3
Coremark	73	0	100	6

Table V presents the results of fault injection in combinational logic. When simulating the occurrence of a single fault, we noticed that the vector addition algorithm obtained an error propagation rate reduced by approximately 60% when the processor is hardened. When used the CCSDS-123 algorithm, the error rate was reduced by 84%, and when used the Coremark, the error rate decreased by 78%. When simulating the injection of 10 faults, the execution of all algorithms obtained a high error propagation rate, even when using the hardened version of the processor. However, the hardened processor was still able to mask some errors, as the error rate in the algorithms reduced between 13% and 21% compared to the processor without fault tolerance techniques.

Among the works that evaluate the efficiency of fault tolerance techniques, the work [6] presents an analysis of fault coverage, while others do not present experimental results. In [6], the authors obtained an average error propagation rate improvement of 11.8% when using the processor with fault tolerance techniques. In this work, we obtained an average

TABLE V
ERROR PROPAGATION FOR SET

Algorithm	1-fault injection		10-fault injection	
	Non-hardened	Hardened	Non-hardened	Hardened
Vector addition	39	16	99	78
CCSDS 123	96	15	100	87
Coremark	72	16	100	87

error propagation rate improvement of 50.4% when injecting 10 SEU or SET faults. This result is because the developed processor does not have as many elements susceptible to faults in comparison to the processor described in [6]. Furthermore, the reliable RISC-V core developed by [19] made a test using a neutrons beam and reported an improvement in the mean work to failure of 24x. In comparison, our simulations estimate a 16x improvement in error propagation considering the Coremark algorithm and a 10-fault injection.

VI. CONCLUSIONS

In this work, we developed an agile RISC-V based processor with the provision of fault tolerance techniques. We protected the components that are the most affected by SEU and SET in the space environment. The control logic and ALU structures were protected using TMR, and all registers were protected using Hamming coding. The proposed processor has a combinational circuit cost similar to other RISC-V processors. Because of its simplicity and the design choices, the processor has a lower occupancy of FFs compared to the alternatives.

Compared to concurrent fault-tolerant RISC-V processors, our implementation achieved a smaller overhead. Providing processor fault tolerance has led to an increase in silicon and power costs. However, in applications that require reliability, this overhead is justified by the improvement in reliability.

As future work, we intent to test the processor at the ISIS Neutron research center using the ChipIr microelectronics irradiation instrument, which allows performing SEE testing with neutron beams. This test will represent the first step of qualification for the processor for use in future computing systems embedded in nano-satellites.

ACKNOWLEDGMENTS

This study was financed by the National Council for Scientific and Technological Development (grants 315287/2018-7 and 436982/2018-8). The authors also thank for the support of the University Space Center of Montpellier, France.

REFERENCES

[1] M. Yang, G. Hua, Y. Feng, and J. Gong, *Fault-tolerance techniques for spacecraft control computers*, 1st ed. Wiley Publishing, 2017.

[2] D. J. Sorin, "Fault tolerant computer architecture," *Synthesis Lectures on Computer Architecture*, vol. 4, no. 1, pp. 1–104, 2009.

[3] M. Hijorth *et al.*, "GR740: Rad-hard quad-core LEON4FT system-on-chip," in *DASIA 2015-DAta Systems in Aerospace*, vol. 732, 2015.

[4] C. A. Hulme, H. H. Loomis, A. A. Ross, and R. Yuan, "Configurable fault-tolerant processor (CFTP) for spacecraft onboard processing," in *2004 IEEE Aerospace Conf. Proc. (IEEE Cat. No.04TH8720)*, vol. 4, March 2004, pp. 2269–2276 Vol.4.

[5] B. Ustaoglu and B. O. Yalcin, "Fault tolerant register file design for MIPS AES-crypto microprocessor," in *2015 IEEE International Conf. on Electronics, Circuits, and Systems (ICECS)*, Dec 2015, pp. 442–445.

[6] M. Didehban, S. Khosjbakht, H. R. Zarandi, and S. Pourmozaffari, "Reducing of soft error effects on a MIPS-based dual-core processor," in *2010 15th CSI International Symp. on Computer Architecture and Digital Systems*, Sep. 2010, pp. 151–152.

[7] A. Waterman, Y. Lee, D. A. Patterson, and K. Asanovic, "The RISC-V instruction set manual, volume i: Base user-level isa," *EECS Dept., UC Berkeley, Tech. Rep. UCB/EECS-2011-62*, vol. 116, 2011.

[8] P. D. Schiavone *et al.*, "Slow and steady wins the race? a comparison of ultra-low-power RISC-V cores for Internet-of-Things applications," in *2017 27th International Symp. on Power and Timing Modeling, Optimization and Simulation (PATMOS)*, Sep. 2017, pp. 1–8.

[9] C. Wolf, "PicoRV32 - a size-optimized RISC-V CPU," 2019. [Online]. Available: https://github.com/cliffordwolf/picorv32

[10] PULP platform, "Ariane documentation 2019," 2019. [Online]. Available: https://pulp-platform.github.io/ariane/docs/home/

[11] C. Celio, P.-F. Chiu, B. Nikolic, D. A. Patterson, and K. Asanović, "BOOMv2: an open-source out-of-order RISC-V core," in *First Wksp. on Computer Architecture Research with RISC-V (CARRV)*, 2017.

[12] A. Traber, M. Gautschi, and P. D. Schiavone, "RI5CY: User manual," *ETH Zürich, Tech. Rep.*, 2019.

[13] K. Asanović *et al.*, "The rocket chip generator," EECS Dept., Univ. of California, Berkeley, Tech. Rep. UCB/EECS-2016-17, Apr 2016. [Online]. Available: http://www2.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-17.html

[14] Vectorblox, "Vectorblox ORCA," 2019. [Online]. Available: https://github.com/VectorBlox/orca

[15] E. Matthews and L. Shannon, "Taiga: A new risc-v soft-processor framework enabling high performance cpu architectural features," in *2017 27th International Conf. on Field Programmable Logic and Applications (FPL)*, Sep. 2017, pp. 1–4.

[16] C. Duran, L. Rueda, G. Castillo, A. Agudelo, C. Rojas, L. Chaparro, H. Hurtado, J. Romero, W. Ramirez, H. Gomez, H. Hernandez, J. Amaya, and E. Roa, "A 32-bit microcontroller featuring a RISC-V core," 2016. [Online]. Available: https://github.com/onchipuis/mriscv

[17] M. Samsoniuk, "Opensource RISC-V DarkRISCV," 2019. [Online]. Available: https://github.com/darklife/darkriscv

[18] W. F. Heida, "Towards a fault tolerant risc-v softcore," Ph.D. dissertation, Delft Univ. of Technology, The address of the publisher, 7 2016, http://resolver.tudelft.nl/uuid:cee5e97b-d023-4e27-8cb6-75522528e62d.

[19] A. E. Wilson and M. Wirthlin, "Neutron radiation testing of fault tolerant risc-v soft processor on xilinx sram-based fpgas," in *2019 IEEE Space Computing Conf. (SCC)*, July 2019, pp. 25–32.

[20] B. Pratt, M. Caffrey, P. Graham, K. Morgan, and M. Wirthlin, "Improving fpga design robustness with partial tmr," in *2006 IEEE International Reliability Physics Symp. Proc.*, March 2006, pp. 226–232.

[21] A. Waterman and K. Asanović, "The RISC-V instruction set manual volume i: Unprivileged ISA," *RISC-V Foundation (June 2019)*, 2019.

[22] D. A. Patterson and J. L. Hennessy, *Computer Organization and Design RISC-V Edition: The Hardware Software Interface*, 1st ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2017.

[23] D. Kanter, "RISC-V offers simple, modular ISA," *The Linley Group MICROPROCESSOR Report (March 2016)*, 2016.

[24] A. Waterman and K. Asanović, "The RISC-V instruction set manual-volume I: User-level isa-document version 2.2," *RISC-V Foundation (May 2017)*, 2017.

[25] R. Travessini, P. R. C. Villa, F. L. Vargas, and E. A. Bezerra, "Processor core profiling for SEU effect analysis," in *2018 IEEE 19th Latin-American Test Symp. (LATS)*, March 2018, pp. 1–6.

[26] RISC-V:Organization, "Gnu toolchain for RISC-V, including GCC," 2019. [Online]. Available: https://github.com/riscv/riscv-gnu-toolchain

[27] L. M. V. Pereira, D. A. Santos, C. A. Zeferino, and D. R. Melo, "A low-cost hardware accelerator for CCSDS 123 predictor in FPGA," in *2019 IEEE International Symp. on Circuits and Systems (ISCAS)*, May 2019, pp. 1–5.

[28] H. Quinn *et al.*, "Using benchmarks for radiation testing of microprocessors and FPGAs," *IEEE Transactions on Nuclear Science*, vol. 62, no. 6, pp. 2547–2554, Dec 2015.