Early Soft Error Reliability Analysis on RISC-V

Nicolas Lodéa, Willian Nunes, Vitor Zanini, Marcos Sartori, Luciano Ost, Ney Calazans, Rafael Garibotti, and César Marcon

***Abstract*—The adoption of RISC-V processors bloomed in recent years, mainly due to its open standard and free instruction set architecture. However, much remains to help software engineers deliver high-reliability and bug-free applications and systems based on RISC-V IP designs. This work proposes an early soft error reliability assessment of a RISC-V processor, extending the previously proposed SOFIA fault injection framework. Results from 850k fault injections show that choosing the compiler flag -O2 to optimize performance causes 96% more Hang failures than -O0. Software engineers must evaluate compilation parameters on a case-by-case basis to find the best balance between performance and reliability. This work helps software engineers develop fault-tolerant RISC-V-based systems and applications more efficiently.**

***Index Terms*—Reliability, RISC-V, Soft Error, Fault Injection.**

I. Introduction

In recent years, the use of embedded systems has grown significantly, particularly in areas such as the Internet of Things and edge computing [1]. This growth implies systems that differ in performance, security, reliability, and power consumption requirements. On the one hand, software development for such systems plays a vital role in performance, energy efficiency, and reliability [2]. On the other hand, open hardware initiatives like the RISC-V architecture [3] allow collaborative development and bring more security to projects. These initiatives arise in the RISC-V project due to its open and royalty-free characteristics, which is common sense since open-source IP companies and regulatory bodies (e.g., governments) can certify the absence of spyware or malware on chips after the manufacturing process [4].

Improving reliability and the security level of embedded systems requires developing mechanisms and environments to assess permanent faults, also known as *hard errors*, or transient faults, known as *soft errors*. While hardware design defects usually cause the first, the second is generally caused by phenomena such as the incidence of radiation, which may cause *single-event upsets* (SEUs) [5]. The occurrence of an SEU can corrupt memory data, impair application output, or even cause a catastrophic system failure, potentially resulting in loss of life in safety-critical applications, such as pedestrian detection algorithms in autonomous vehicles. Such faults have motivated the increase in research on the reliability of embedded algorithms [6]–[10].

N. Lodéa, W. Nunes, V. Zanini, M. Sartori, N. Calazans, R. Garibotti and C. Marcon are with the School of Technology, PUCRS, Brazil (e-mail: {nicolas.lodea, willian.nunes, vitor.balbinot, marcos.sartori}@edu.pucrs.br, {ney.calazans, rafael.garibotti, cesar.marcon}@pucrs.br).

L. Ost is with the Wolfson School, Loughborough University, UK (e-mail: l.ost@lboro.ac.uk).

Embedded systems reliability exploration imposes significant challenges, including: (*i*) the ability to perform a large number of tests in a reasonable time; (*ii*) providing engineers with a detailed view of the system behavior in the presence of failures; and (*iii*) identify associations between application characteristics and processor-specific parameters in large datasets resulting from the tests. Exposing these systems to radiation effects results in more accurate data. However, the high cost and time of planning and verification make this assessment impractical for most projects. Thus, fault injection techniques are often a better method to evaluate soft errors in embedded systems at project development time [11]. Simulation and emulation tools typically inject faults using a processor model, usually at high or medium abstraction levels.

The current scenario requires faster and more efficient means to assess the embedded system reliability. Frameworks based on virtual platforms (VPs) achieved popularity in academia and many industrial sectors over the years due to their design flexibility, debug capability, and simulation performance. For example, Akram *et al.* [12] describe a gem5 extension to simulate reliable execution environments. Other recent works incorporate fault injection capabilities into VP frameworks [13], [14], allowing the analysis of complex applications and different processor architectures in the early design phases. The main contribution of this paper is the extension of a VP framework to support fault injection in RISC-V processors. The work contributes to the use of VPs in the reliability assessment during the early design phases of RISC-V-based systems. It enables software engineers or other professionals involved in the design to develop fault-tolerant RISC-V-based systems and applications more efficiently. Other contributions of the paper are: (*i*) the presentation of *PUCRS-RV* [15], an open-source synthesizable implementation of the RISC-V processor with support for the RV32I instruction set; and (*ii*) the reliability investigation of the SEU occurrence on RISC-V processors.

The rest of this paper is organized as follows. Section II presents related works regarding soft error assessment of RISC-V processors and the different fault injection environments found in the literature. Section III presents the PUCRS-RV processor and the SOFIA VP framework. Furthermore, it details the proposed extension and this work's adopted fault classification. Section IV assesses the SEU reliability of the PUCRS-RV processor. Finally, Section V presents the final remarks and future work.

II. Related Works

The increasing complexity of computer systems and the need to assess their reliability with applications of different

TABLE I
RELATED WORKS ON THE RELIABILITY OF RISC-V PROCESSORS WITH DIFFERENT FAULT INJECTION ABSTRACTION LEVELS.

Works	Year	Processor RISC-V	Processor Arm	FI Abstraction Level	Fault type	Number of Fault Injections	Applications
Wilson and Wirthlin [16]	2019	✓		FPGA (Radiation)	SEU	—	Dhrystone
Oliveira *et al.* [17]	2020	✓		FPGA (Radiation), FPGA (Emulation)	SEE	—	Matrix Mult., Quicksort, AES
Ramos *et al.* [18]	2017	✓		FPGA (Emulation)	SEU	10k	Quicksort, Matrix Mult., Tower of Hanoi, Dijkstra, Mergesort, FFT
Cho [19]	2018	✓		FPGA (Emulation)	SEMU	80k	SPECINT 2000, MinneSPEC
Ramos *et al.* [20]	2019	✓		FPGA (Emulation)	SEU	10k	Dijkstra, Fibonacci, Matrix Mult., Quicksort
Wali *et al.* [21]	2020	✓		FPGA (Emulation)	SEE (SEFI)	10k	Tower of Hanoi, Dijkstra, Matrix Mult., Quicksort, Merge Sort
Marques *et al.* [22]	2021	✓		FPGA (Emulation)	SEU	100k	Fibonacci
Mohseni and Reviriego [23]	2019	✓		FPGA (Emulation), RTL (Simulation)	SEU	10k	Bubble Sort, Dijkstra, FFT, Matrix Mult., Quadratic Equation
Gupta *et al.* [24]	2015	✓		RTL (Simulation)	SEE	—	In-house Application
Santos *et al.* [25]	2020	✓		RTL (Simulation)	SEU	100	Array Sum, CCSDS, Coremark
Bandeira *et al.* [26]	2019		✓	VP (Simulation)	SEU	800	YOLOv3 algorithm
Abich *et al.* [14]	2021		✓	VP (Simulation)	SEU	1k a 10k	52 applications
This work	**2022**	✓		**ASIC (Synthesis), RTL and VP (Simulation)**	**SEU**	**17k**	**25 applications**

criticality levels drive research to describe and evaluate the functionality of these systems at high levels of abstraction. This section reviews the literature focusing on the RISC-V processor and environments to assess its reliability, comparing it with the approach proposed herein.

Table I summarizes the state-of-the-art RISC-V processor reliability and the related test environments, covering different fault injection abstraction levels. Except for references [14], [26], and the present work, none of the reviewed approaches addresses VPs. Most works assess the reliability of RISC-V processors with a high level of accuracy through descriptions of low-to-medium abstraction levels. This includes works with radiation testing [16], [17], FPGA emulation [17]–[23] and RTL simulation [23]–[25]. The choices have high financial costs (radiation testing) or are more time-consuming (FPGA emulation and RTL simulation), compared to VP-based test environments. VP-based fault injection environments, in turn, present a larger degree of imprecision in results. However, Abich *et al.* [14] show that it is possible to control such inaccuracies, keeping them at or below 10% if coupled with a well-characterized model.

Wilson and Wirthlin [16] and Oliveira *et al.* [17] apply triple modular redundancy (TMR) and scrubbing techniques to protect RISC-V processors against radiation incidence. Radiation testing is justifiable by the need to assess the reliability of RISC-V-based products. Nonetheless, these works evaluate just a few applications and focus on the end of the processor design process. Accordingly, a high-level abstraction approach is best recommended for the early stages of processor design to complement and enhance radiation results.

Works [17]–[23] evaluate the reliability of RISC-V processors emulated in FPGAs with different purposes. Mohseni and Reviriego [23] divide the processor into modules to analyze the individual soft error sensitivity. Wali *et al.* [21] investigate the impact that the Linux operating system causes on the reliability of the lowRISC processor embedded in an FPGA. Cho [19] assesses the reliability of two RISC-V implementations: in-order and out-of-order execution organizations. Results show a lower number of processor failures in the out-of-order organization. However, the fault types of the two processors remain proportional.

Other works propose fault-tolerant architectures emulated in FPGA. Oliveira *et al.* [17] protect the RISC-V processor through different TMR versions. Ramos *et al.* [20] were more specific, protecting only the Arithmetic and Logical Unit (ALU) by generating an ALU version with TMR for the most executed instructions of each tested application. Marques *et al.* [22] propose a fault-tolerant architecture that uses two distinct processors (RISC-V and Arm) to detect and correct faults. The Authors applied the dual-core lockstep (DCLS) technique followed by a return to a recovery point in case of inconsistency between the two processor results.

Works based on RTL simulation focus on processor observability. For example, Mohseni and Reviriego [23] analyze the fault tolerance of each RISC-V module. Gupta *et al.* [24] apply error correction codes (ECCs) to protect registers and memories and spatial-temporal techniques (i.e., dual modular redundancy - DMR and delays) to protect the ALU. Then, they quantify how these techniques help the architecture continue operating even with a certain degree of degradation by applying transient and permanent faults. With the same fault tolerance bias, Santos *et al.* [25] implement TMR to protect the ALU and Hamming code to protect the memory.

Regarding VP-based works, Abich *et al.* [14] analyze the quality of results from two high-level abstraction fault injection environments: OVPsim and gem5. They recommend using these FI environments early in the design phase as they are 1000× faster than cycle-accurate simulators (i.e., RTL level) and up to 312× faster than dedicated simulators (e.g., gem5). They also preserve more than 90% accuracy in the analysis of SEU faults in computer systems. Bandeira *et al.* [26] propose a non-intrusive fault injection environment for fast reliability assessment of SEU-type faults. Authors validate the environment through an automotive case study executing up to 43 billion instructions. Results show that the isolation of critical application functions can lead to a more efficient error analysis, reducing masked faults by 28%.

The present work stands out against the others in Table I

as the first to explore the reliability of the RISC-V processor using fault injection through a virtual platform. Another original contribution is a large number of applications (i.e., 25) and the number of faults injected in the experiments (a total of 850,000 faults) compared to the works reported in Table I, ensuring a broad coverage and high statistical relevance to results. Additionally, RTL simulation is employed as a reference, and a synthesizable RISC-V version is provided for future comparisons and explorations.

III. Fault Injection Methodology

This section describes: (*i*) the original SOFIA framework; (*ii*) the SOFIA's extension to support the RISC-V processor; (*iii*) the PUCRS-RV, an open-source implementation of the RISC-V processor with support for the RV32I instruction set; and (*iv*) the adopted fault injection classification.

A. SOFIA Framework

The SOFIA framework [26] provides fault injection methods that emulate single bit upsets (SBUs) by injecting faults into different system parts such as registers, physical memory, application virtual memory, function calls, application variables, and data structures. Another remarkable feature is the possibility of isolating specific parts of the application's runtime, allowing individual and dynamic analysis of the executed instructions. In its previous version, SOFIA only supported ARM architectures, the present work extends this support to RISC-V processors.

Fig. 1 illustrates the five phases of the SOFIA framework. Phase 1 compiles the application's source code and generates the object file. Phase 2 is the flawless execution, where the application is simulated to verify its correctness and reference information is extracted (i.e. register context and final memory state). SOFIA may also acquire additional information during this phase depending on the fault injection technique selected (e.g. functions and variable addresses). Phase 3 calls a SOFIA module that generates a list containing the fault injection configurations, i.e. fault injection time, register name, and target bit for each fault injection experiment. In SOFIA, the FI configuration is based on a uniform random function, a well-accepted fault injection technique that covers most faults in a system with low computational cost [27]. In Phase 4, the SOFIA FI module executes the fault experiments, by reading the fault characteristics and inserting the inverted bit according to the information provided in Phase 3. After each execution, another SOFIA module compares its result (e.g. the number of instructions executed, the register context, and the memory state) with the flawless execution and generates an individual report. Finally, Phase 5 gathers all the individual reports (i.e. reliability results from each experiment) into a single file. Then, SOFIA performs several statistical analyses (e.g. percentage of fault types, worst and best cases) and produces individual and collective graphs that help engineers understand the soft error reliability of the underlying architecture.

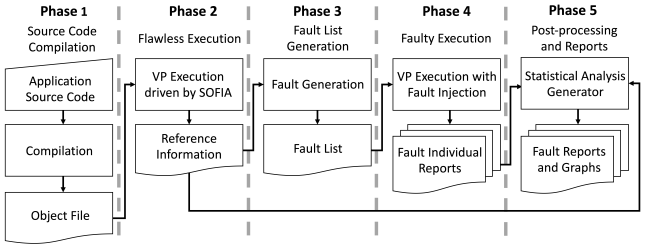


Fig 1. The five phases of the SOFIA framework.

B. Proposed SOFIA Extension

The first phases of the original SOFIA framework flow were modified to support the reliability assessment of transient faults in RISC-V processors. This work selected the Multicore Developer (M*DEV) virtual platform [28] as reference. In so doing, SOFIA can now use all processor models supported by M*DEV and their execution reports.

In Phase 1, the *Makefile* was adapted to support the compilation of any version of the RISC-V processor ISA options. In Phase 2, SOFIA manages the flawless execution in the M*DEV. Originally, the SOFIA framework supported only virtual platforms using Arm processor models. Thus, the *script tcl* (referring to the platform configuration) has been rewritten to be generic, providing support to RISC-V processors and paving the way for new processor models. The result is an application execution report on the RISC-V processor (i.e. reference information), as Fig. 1 shows. In addition to these modifications, SOFIA must contain a description of the target processor registers to generate the list with the fault injection configurations. This description refers to the naming convention used by RISC-V registers as specified in the application binary interface (ABI). For example, this work uses the RV32I instruction set, which assumes an organization with 32 general-purpose registers and the *program counter* (pc) that may be faulty. Therefore, the *scripts* have been changed to support this new register list, renamed as follows: *ra*, *sp*, *gp*, *tp*, *t0 – t6*, *s0 – s11*, *a0 – a7*, and *pc*. This set of modifications allowed injecting faults into any available register of the RISC-V processor.

C. PUCRS-RV

The PUCRS-RV processor is a synchronous organization of the RISC-V architecture, designed as a 5-stage pipeline, in-order processor, with speculative capability. The RISC-V processor derived from an asynchronous organization first described by Sartori *et al.* [29] and implements the RV32I instruction set, which is equivalent to the model available on the M*DEV virtual platform [28].

Fig. 2 shows the block diagram of the PUCRS-RV processor. Later on, there is a description of some PUCRS-RV details that facilitate replication on other virtual platforms (e.g. gem5).

Stage 1 is responsible for *Instruction Fetch* from memory. The fetch unit identifies the stream and manages the *pc* register, assuming the value of the subsequent memory position (i.e. *pc*+4) or the address of a jump taken. The jump prediction policy is "never jump". Note that each instruction that leaves Stage 1 is associated with a *tag* that will follow the instruction

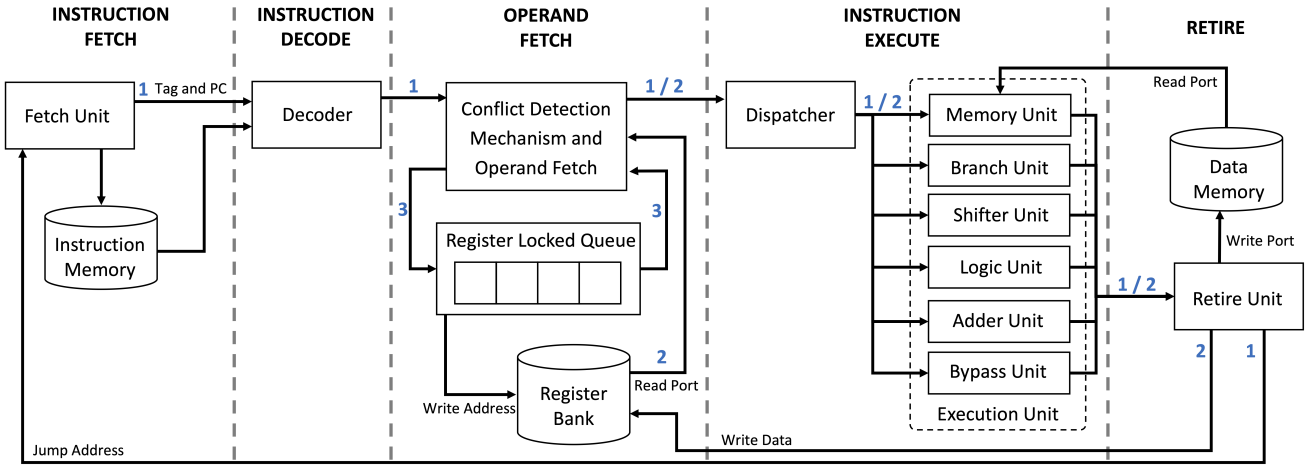


Fig 2. Block diagram of the PUCRS-RV processor, highlighting in blue the three processor execution loops.

to the last stage for validation. Then, the *tag* is incremented whenever a jump is performed, signaling that the instruction belongs to a new context.

Stage 2 is *Instruction Decode*, responsible for generating control signals based on the object code of the instruction fetched in Stage 1. Stage 3 is *Operand Fetch* and contains the conflict detection mechanism (i.e. *hazards*) that employs a queue of locked registers. If there are no conflicts, operands are fetched from the register bank and propagated to the next stage. Otherwise, bubbles (equivalent to *NOP* instructions) are generated until the conflict is resolved by writing to the locked register. Stage 4 is *Instruction Execute* and incorporates a Dispatcher that sends operands only to the unit responsible for that type of instruction.

The last stage of the pipeline is *Retirement*. The *Retirement Unit* is responsible for removing the instruction from the processor and closing the pipeline loops. The *tag* that accompanies the instruction is validated in this unit by comparing it with the unit's internal *tag*. If the *tag* matches, the instruction can be executed. Otherwise, this indicates that they belong to different contexts, i.e. that a jump was executed after fetching that instruction, and its effects should be discarded.

Fig. 2 also illustrates the three loops of the RISC-V processor execution flow. The *1st loop* (label 1) controls the overall execution flow. The *2nd loop* (label 2) comprises the 3rd to the 5th stage of the pipeline and is a data loop. It includes the fetching of operands from the register bank, the instruction execution, and the last stage, which is eventually written to the register bank. Finally, the *3rd loop* (label 3) comprises the mechanism for identifying data conflicts. If any instruction entering the third stage has data coming from a locked register, a conflict is identified, and bubbles are inserted until the conflict resolves.

D. Fault Classification

This work adopts the fault classification proposed by Cho *et al.* [11], which defines five possible behaviors for a system in the presence of soft errors:

- *Vanish*: the output matches the expected result, and no trace of the fault is left in the memory system or architectural state, i.e. the introduced fault is fully masked;
- *Output Not Affected* (*ONA*): the application output is flawless. However, one or more bits of the architectural state are incorrect;
- *Output Memory Mismatch* (*OMM*): the application terminates without any indication of failure, but with a resulting incorrect memory state;
- *Unexpected Termination* (*UT*): the application terminates abnormally with an error indication;
- *Hang*: the application does not finish within a margin of 20% beyond the reference execution time.

IV. Results

A. Experimental Setup

One of the main concerns in assessing a processor's soft error reliability is having a realistic and broad coverage approach to fault injection. This work seeks to ensure that the number of fault injections has statistical relevance [30] by applying the equations developed by Leveugle *et al.* [31]. Each test campaign consists of 17,000 experiments with one bit-flip per application. According to Leveugle's equation, results present a margin of error of 1% with a confidence level of 99%. On the one hand, the confidence level guarantees that the same results will be obtained if we repeat the experiments. On the other hand, the margin of error indicates the percentage difference between the results obtained and the expected value of the soft error reliability. Furthermore, our results present greater statistical relevance than most literature studies (Table I).

To ensure complete stimulus coverage for the RISC-V processor, both for computationally demanding and memory intensive algorithms, this work evaluated the soft error reliability of the following 25 applications from the Mälardalen WCET benchmark suit [32]: (A) peakspeed, (B) fibonacci, (C) insert_sort, (D) binary_search, (E) crc, (F) bubble, (G) usqrt, (H) compress, (I) jfdct_int, (J) bit_manipulation, (K) petri_net, (L) switch_cases, (M) blowfish, (N) harm, (O) counts, (P) mdc, (Q) hanoi, (R) expint, (S) factorial, (T) ud, (U) matrix_mult, (V) edn, (W) adpcm,(X) fdct, and (Y) prime.

B. Simulation Performance Analysis

Similar to the analysis presented by Abich *et al.* [14], VP-based fault injection campaigns can be 1000× faster than

TABLE II
ASSESSMENT OF FAULT INJECTION CAMPAIGNS DETAILING THE CHARACTERISTICS OF EACH APPLICATION (NUMBER OF REGISTERS, SIMULATION TIME AND NUMBER OF INSTRUCTIONS); APPLICATION PROFILE (PERCENTAGE OF ARITHMETIC INSTRUCTIONS, UNCONDITIONAL INSTRUCTIONS, CONDITIONAL INSTRUCTIONS AND MEMORY INSTRUCTIONS); AND APPLICATION RELIABILITY (I.E., VANISH, ONA, OMM, UT AND HANG).

App	Application Characteristics						Application Profile								Application Reliability									
	# Reg.		S. Time (*ks*)		# Inst.		Arit.		Uncond.		Cond.		Memory		Vanish		ONA		OMM		UT		Hang	
	O0	O2	O0	O2	O0	O2	O0	O2	O0	O2	O0	O2	O0	O2	O0	O2	O0	O2	O0	O2	O0	O2	O0	O2
A	22	23	1.13	0.86	19k	5k	5.8	11.4	0.3	1.2	2.8	10.1	91.1	77.3	59.7	54.0	17.3	15.0	19.0	26.5	2.3	2.4	1.7	2.1
B	22	22	1.21	0.88	266k	132k	21.8	28.8	0.2	0.1	7.3	14.4	70.7	56.7	59.7	65.0	18.0	18.2	18.4	11.6	2.2	3.4	1.7	1.8
C	22	23	1.23	0.87	221k	66k	38.9	15.2	4.5	13.6	9.1	28.7	47.5	42.5	59.6	53.4	18.4	18.2	18.2	20.1	2.3	2.5	1.5	5.8
D	22	22	1.28	0.86	136k	40k	31.4	49.8	4.3	2.4	12.2	36.2	52.1	11.6	60.2	55.9	18.4	20.8	16.7	19.0	2.5	2.3	2.2	2.0
E	22	23	1.38	0.92	188k	25k	35.7	70.9	3.0	0.3	17.4	17.6	43.9	11.2	59.8	57.9	18.3	14.4	17.3	22.2	2.3	2.4	2.3	3.1
F	22	22	1.41	0.91	225k	35k	39.9	15.2	0.1	0.2	4.5	28.3	55.5	56.3	58.5	53.1	17.8	18.0	18.8	19.2	2.6	2.1	2.3	7.6
G	22	22	1.47	0.95	194k	75k	34.3	77.6	0.6	0.6	7.3	18.4	57.8	3.4	61.0	56.5	18.2	19.9	17.4	16.0	1.4	2.1	2.0	5.5
H	22	28	1.64	1.23	186k	116k	47.4	18.2	1.6	1.3	5.5	8.6	45.5	71.9	68.8	76.9	17.5	13.0	7.3	3.0	1.9	2.2	4.5	4.9
I	22	30	1.72	1.12	339k	182k	51.6	80.0	0.2	0.3	3.0	5.4	45.2	14.3	58.3	69.8	17.8	1.8	19.3	21.1	2.6	2.3	2.0	5.0
J	22	27	1.76	1.14	304k	112k	36.4	60.2	2.5	0.6	5.7	14.8	55.4	24.4	61.8	69.9	17.9	12.9	15.9	7.6	2.6	2.7	1.8	6.9
K	23	27	1.94	1.42	149k	90k	28.3	0.2	0.1	10.5	21.0	34.4	50.6	54.9	59.6	59.5	18.0	18.2	19.7	17.7	2.0	2.4	0.7	2.2
L	22	22	1.95	0.92	366k	70k	34.8	49.1	10.1	1.0	9.9	48.6	45.2	1.3	59.4	54.3	17.0	17.9	18.6	18.4	2.5	2.2	2.5	7.2
M	22	26	1.99	1.30	904k	276k	40.6	74.7	2.0	0.4	1.2	3.7	56.2	21.2	61.5	55.9	15.7	9.6	18.5	22.9	3.1	2.9	1.2	8.7
N	22	22	2.00	1.05	509k	249k	28.0	35.7	9.9	9.6	16.8	34.4	45.3	20.3	62.3	61.4	18.4	19.9	14.5	6.7	1.8	1.9	3.0	10.1
O	22	23	2.07	1.60	445k	274k	46.4	48.7	4.9	4.6	22.6	36.5	26.1	10.2	60.6	62.3	17.0	14.8	17.2	13.5	2.6	2.9	2.6	6.5
P	22	22	2.28	1.12	570k	256k	32.4	33.9	10.1	9.5	26.7	34.3	30.8	22.3	59.0	58.7	19.5	22.9	16.6	9.4	2.5	3.0	2.4	6.0
Q	22	22	2.28	1.55	754k	368k	22.7	15.6	9.1	11.1	4.6	8.9	63.6	64.4	62.1	65.9	17.5	18.3	15.6	9.6	1.4	1.5	3.4	4.7
R	22	22	2.38	1.58	652k	459k	26.4	33.1	11.6	16.2	19.7	27.1	42.3	23.6	60.9	66.3	18.0	20.9	17.1	4.7	1.9	2.2	2.1	5.9
S	22	22	2.42	0.87	734k	4k	37.0	31.9	9.4	1.3	17.6	31.8	36.0	35.0	60.8	56.6	19.4	19.2	16.0	18.6	1.6	2.1	2.2	3.5
T	22	26	2.52	1.17	744k	265k	47.5	43.8	2.9	6.9	10.1	23.2	39.5	26.1	58.9	67.7	19.7	16.0	17.5	4.5	1.6	2.2	2.3	9.6
U	22	22	2.53	1.91	1.0M	685k	55.4	57.4	2.0	2.8	20.8	32.6	21.8	7.2	58.6	56.8	17.2	17.8	20.1	12.4	1.9	2.7	2.2	10.3
V	23	29	2.98	2.42	1.0M	819k	56.6	61.0	1.4	1.6	29.6	33.5	12.4	3.9	60.1	72.9	17.0	6.3	18.5	12.9	2.4	2.6	2.0	5.3
W	22	25	6.64	4.12	2.8M	1.7M	56.6	57.2	2.3	2.6	32.6	35.6	8.5	4.6	60.3	62.3	16.6	13.4	18.9	19.7	2.1	2.4	2.1	2.2
X	22	30	7.00	1.37	3.0M	291k	57.6	76.0	1.3	0.1	28.6	4.3	12.5	19.6	57.2	75.2	18.5	1.6	19.9	17.9	2.5	2.4	1.9	2.9
Y	22	22	7.85	7.48	3.4M	2.9M	47.7	52.0	4.1	3.5	34.0	38.2	14.2	6.3	59.8	58.3	19.9	23.6	15.5	12.6	2.5	3.0	2.3	2.5

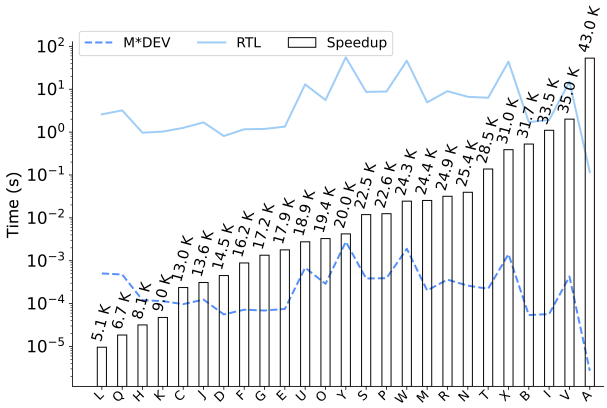


Fig 3. Simulation time of the RISC-V processor on M*DEV and RTL level.

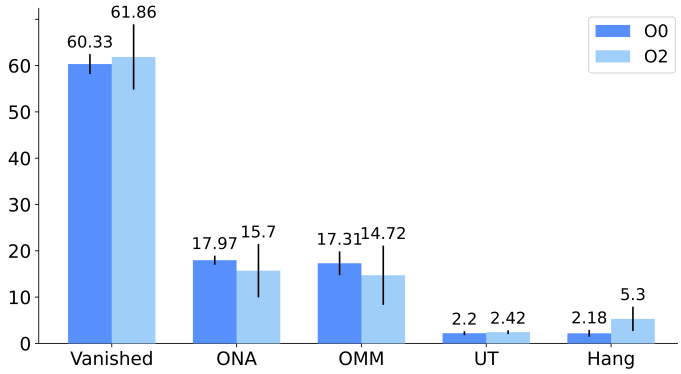


Fig 4. Mean and covariance of faults of the application set.

cycle-accurate simulator campaigns. At the same time, they preserve the soft error reliability accuracy between different simulator abstractions with a mismatch below 10%. Fig. 3 shows the simulation times on the M*DEV virtual platform [28] and the RTL level of the RISC-V processor. The results corroborate those presented in [14] and show superior simulation performance using the RISC-V processor compared to the Arm architecture.

C. RISC-V Processor Reliability Analysis

In the initial design phase, software engineers rely on compiler parameters for specific purposes, e.g., $-O0$ to produce the original design without optimizations or $-O2$ to improve application performance. However, there is no compiler parameter to improve application reliability, nor is it known how much a compiler parameter affects application reliability.

Fig. 4 shows the mean and covariance of faults of the application set using two compiler parameters ($-O0$ and $-O2$). Note that we chose the compiler parameter $-O2$ instead of $-O3$ because its optimizations focus on improving performance without significantly increasing code size, making it more suitable for the Mälardalen WCET benchmark algorithms [32]. Fig. 4 shows that applications' reliability does not significantly vary between the two compiler parameters. It is only observed that the parameter $-O2$ causes more *Hangs*. On the other hand, it is known that each application has different behavior and the compiler's optimization parameter choice directly reflects on the reliability of the entire system.

Table II shows the results of the fault injection campaigns for the 25 applications using two compiler parameters ($-O0$ and $-O2$). It presents the uniqueness of each application. For example, application *C* showed better soft error reliability with $-O0$ since it had an increased percentage of *Vanish* and a decrease in other fault types. On the other hand, application *H* showed better soft error reliability with $-O2$.

Due to the different applications' characteristics, Table II shows the influence that the compiler parameter has on the application performance and the memory footprint used by

the object code, where the $-O2$ parameter employs more registers to provide better performance to the application. Furthermore, the compiler's optimization parameter choice impacts the change in the generated application profile, changing its susceptibility to soft errors.

Table II shows how the compiler parameters affect the application profile. On the one hand, choosing the $-O2$ parameter over $-O0$ increases the percentage of conditional instructions, the only exception being application *X*. On the other hand, in more than 80% of cases, this choice also decreases the percentage of memory instructions, and in 77% of cases increases arithmetic instructions.

The soft error reliability analysis shown in Table II presents that application H has the best reliability on the RISC-V processor using the two compiler parameters ($-O0$ and $-O2$). One indication is the low percentage of *OMM* in both cases. Furthermore, the *UT* has low variation among all applications, indicating that the susceptibility to soft errors is more related to the RISC-V processor architecture than the application profile.

On the other hand, reliability-specific results show that *N*, *T*, and *U* applications present the highest percentages of *Hangs* (around 10%) caused by the $-O2$ parameter, which were influenced by the increase in conditional instructions and the decrease in memory instructions.

Another factor that causes reduced reliability is shown in cases with a low percentage of *Vanish*. These applications generally increase the arithmetic instructions and decrease the memory instructions. It suggests that a register that had a bit flip may have been used in computations (i.e., arithmetic operations) before being stored in memory, making it easier for the fault to become a failure in the RISC-V processor. The exceptions are *C*, *F*, *K*, and *S* applications. In these cases, the increase in conditional instructions outweighs the rise in arithmetic instructions, suggesting that faulty registers directly affect the application's selection and control, making these applications less fault-tolerant.

These results show the need to analyze case by case. Therefore, extending a fault injection environment in the early design phases of RISC-V-based projects is imperative. This environment helps software engineers search for more fault-tolerant applications.

V. Conclusions

This work extended the SOFIA framework and investigated the soft error reliability of 25 applications running on a RISC-V processor. Evaluations using different compiler parameters show that if, on the one hand, performance-oriented parameters ($-O2$) cause more faults of *Hang* type. On the other hand, software engineers must evaluate the compiler parameters on a case-by-case basis to find the best balance between performance and reliability because different application characteristics affect processor reliability differently. Furthermore, PUCRS-RV is presented here, an open-source synthesizable implementation of the RISC-V processor supporting the RV32I instruction set. The SOFIA framework characterized this processor to help software engineers develop designs based on fault-tolerant RISC-V processors.

Acknowledgment

This work was partially funded by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001, CNPq (grants no. 312917/2018-0, no. 309762/2020-0 and no. 317087/2021-5), and FAPERGS.

References

[1] Z. Zou, Y. Jin, P. Nevalainen, Y. Huan, J. Heikkonen, and T. Westerlund, "Edge and Fog Computing Enabled AI for IoT - An Overview," in *AICAS*, 2019, pp. 51–56.

[2] P.-K. Huang and S. Ghiasi, "Power-Aware Compilation for Embedded Processors with Dynamic Voltage Scaling and Adaptive Body Biasing Capabilities," in *DATE*, 2006, pp. 1–2.

[3] A. Waterman, Y. Lee, D. A. Patterson, and K. Asanović, "The RISC-V Instruction Set Manual, Volume I: UserLevel ISA, Version 2.1," UCB/EECS-2016-118, UC Berkeley, Tech. Rep., 2016.

[4] S. Greengard, "Will RISC-V Revolutionize Computing?" *Communications of the ACM*, vol. 63, no. 5, pp. 30–32, 2020.

[5] M. Snir, R. W. Wisniewski, J. A. Abraham, S. V. Adve, S. Bagchi, P. Balaji, J. Belak, P. Bose, F. Cappello, B. Carlson, A. A. Chien, P. Coteus, N. A. Debardeleben, P. C. Diniz, C. Engelmann, M. Erez, S. Fazzari, A. Geist, R. Gupta, F. Johnson, S. Krishnamoorthy, S. Leyffer, D. Liberty, S. Mitra, T. Munson, R. Schreiber, J. Stearley, and E. V. Hensbergen, "Addressing Failures in Exascale Computing," *International Journal of High Performance Computing Applications*, vol. 28, no. 2, pp. 129–173, 2014.

[6] F. R. da Rosa, R. Garibotti, L. Ost, and R. Reis, "Using Machine Learning Techniques to Evaluate Multicore Soft Error Reliability," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 66, no. 6, pp. 2151–2164, 2019.

[7] M. G. Trindade, R. P. Bastos, R. Garibotti, L. Ost, M. Letiche, and J. Beaucour, "Assessment of Machine Learning Algorithms for Near-Sensor Computing under Radiation Soft Errors," in *ICECS*, 2020, pp. 1–4.

[8] V. Bandeira, J. Sampford, R. Garibotti, M. G. Trindade, R. P. Bastos, R. Reis, and L. Ost, "Impact of radiation-induced soft error on embedded cryptography algorithms," *Microelectronics Reliability*, p. 114349, 2021.

[9] G. Abich, J. Gava, R. Garibotti, R. Reis, and L. Ost, "Applying Lightweight Soft Error Mitigation Techniques to Embedded Mixed Precision Deep Neural Networks," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 68, no. 11, pp. 4772–4782, 2021.

[10] G. Abich, R. Garibotti, R. Reis, and L. Ost, "The Impact of Soft Errors in Memory Units of Edge Devices Executing Convolutional Neural Networks," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 69, no. 3, pp. 679–683, 2022.

[11] H. Cho, S. Mirkhani, C.-Y. Cher, J. A. Abraham, and S. Mitra, "Quantitative Evaluation of Soft Error Injection Techniques for Robust System Design," in *DAC*, 2013, pp. 1–10.

[12] A. Akram, V. Akella, S. Peisert, and J. Lowe-Power, "Enabling Design Space Exploration for RISC-V Secure Compute Environments," *Berkeley Lab.*, 2022. [Online]. Available: https://escholarship.org/uc/item/0nt7h5jm

[13] K. Parasyris, G. Tziantzoulis, C. D. Antonopoulos, and N. Bellas, "GemFI: A Fault Injection Tool for Studying the Behavior of Applications on Unreliable Substrates," in *DSN*, 2014, pp. 622–629.

[14] G. Abich, R. Garibotti, V. Bandeira, F. Rosa, J. Gava, F. Bortolon, G. Medeiros, F. G. Moraes, R. Reis, and L. Ost, "Evaluation of the soft error assessment consistency of a JIT-based virtual platform simulator," *IET Computers & Digital Techniques*, vol. 15, no. 2, pp. 125–142, 2021.

[15]

[16] A. E. Wilson and M. Wirthlin, "Neutron Radiation Testing of Fault Tolerant RISC-V Soft Processor on Xilinx SRAM-based FPGAs," in *SCC*, 2019, pp. 25–32.

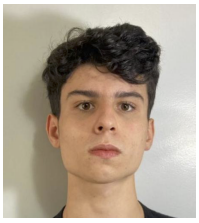
[17] A. B. de Oliveira, L. A. Tambara, F. Benevenuti, L. A. C. Benites, N. Added, V. A. P. Aguiar, N. H. Medina, M. A. G. Silveira, and F. L. Kastensmidt, "Evaluating Soft Core RISC-V Processor in SRAM-Based FPGA Under Radiation Effects," *IEEE Transactions on Nuclear Science*, vol. 67, no. 7, pp. 1503–1510, 2020.

[18] A. Ramos, J. A. Maestro, and P. Reviriego, "Characterizing a RISC-V SRAM-based FPGA implementation against Single Event Upsets using fault injection," *Microelectronics Reliability*, vol. 78, pp. 205–211, 2017.

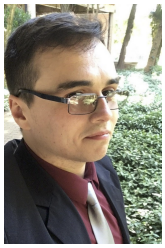
[19] H. Cho, "Impact of Microarchitectural Differences of RISC-V Processor Cores on Soft Error Effects," *IEEE Access*, vol. 6, pp. 41 302–41 313, 2018.
[20] A. Ramos, R. G. Toral, P. Reviriego, and J. A. Maestro, "An ALU Protection Methodology for Soft Processors on SRAM-Based FPGAs," *IEEE Transactions on Computers*, vol. 68, no. 9, pp. 1404–1410, 2019.
[21] I. Wali, A. Sánchez-Macián, A. Ramos, and J. A. Maestro, "Analyzing the impact of the Operating System on the Reliability of a RISC-V FPGA Implementation," in *ICECS*, 2020, pp. 1–4.
[22] I. Marques, C. Rodrigues, A. Tavares, S. Pinto, and T. Gomes, "Lock-V: A heterogeneous fault tolerance architecture based on Arm and RISC-V," *Microelectronics Reliability*, vol. 120, p. 114120, 2021.
[23] Z. Mohseni and P. Reviriego, "Reliability characterization and activity analysis of lowRISC internal modules against single event upsets using fault injection and RTL simulation," *Microprocessors and Microsystems*, vol. 71, p. 102871, 2019.
[24] S. Gupta, N. Gala, G. S. Madhusudan, and V. Kamakoti, "SHAKTI-F: A Fault Tolerant Microprocessor Architecture," in *ATS*, 2015, pp. 163–168.
[25] D. A. Santos, L. M. Luza, C. A. Zeferino, L. Dilillo, and D. R. Melo, "A Low-Cost Fault-Tolerant RISC-V Processor for Space Systems," in *DTIS*, 2020, pp. 1–5.
[26] V. Bandeira, F. Rosa, R. Reis, and L. Ost, "Non-intrusive Fault Injection Techniques for Efficient Soft Error Vulnerability Analysis," in *VLSI-SoC*, 2019, pp. 123–128.
[27] S. Feng, S. Gupta, A. Ansari, and S. Mahlke, "Shoestring: Probabilistic Soft Error Reliability on the Cheap," in *ASPLOS*, 2010, pp. 385–396.
[28] Imperas, "DEV - Virtual Platform Development and Simulation," 2022. [Online]. Available: https://www.imperas.com/dev-virtual-platform-development-and-simulation
[29] M. L. L. Sartori and N. L. V. Calazans, "Go Functional Model for a RISC-V Asynchronous Organisation - ARV," in *ICECS*, 2017, pp. 381–348.
[30] M. Krzywinski and N. Altman, "Points of significance: Significance, P values and t-tests," *Nature Methods*, vol. 10, no. 11, pp. 1041–1042, 2013.
[31] R. Leveugle, A. Calvez, P. Maistri, and P. Vanhauwaert, "Statistical Fault Injection: Quantified Error and Confidence," in *DATE*, 2009, pp. 502–506.
[32] J. Gustafsson, A. Betts, A. Ermedahl, and B. Lisper, "The mälardalen WCET benchmarks: past, present and future," in *WCET*, 2010, pp. 136–146.

Nicolas Lodéa received a B.Sc. degree in Computer Engineering in 2019 from the University of Passo Fundo, Brazil. Currently, he is an M.Sc. student at the PPGCC at the PUCRS University, Brazil. His research focus is on soft error simulation with virtual platforms.

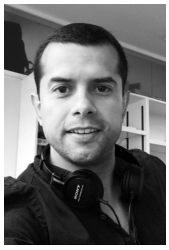
Willian Nunes is an undergraduate student of Computer Engineering at PUCRS. He has participated in undergraduate research programs funded by government agencies. His research interests are in the areas of computer architecture, Microelectronics and non-synchronous circuits.



Vitor Zanini is an undergraduate student of Computer Engineering at PUCRS. He participates in a scientific initiation program at the Autonomous Systems Laboratory. His research interests are in the areas of computer architecture, robotics and embedded systems.

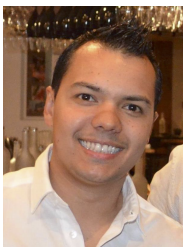


Marcos L. L. Sartori received a B.S. degree in Computer Engineering (2017) and an M.Sc. degree in Computer Science (2019) from PUCRS, Brazil. He is currently pursuing a Ph.D. in Computer Science at the same institution. His research interests include non-synchronous circuits, computer architecture, EDA techniques and tools. He is a Student Member of the IEEE and of the Brazilian Computer Society (SBC).



Luciano Ost is currently a Faculty Member with Loughborough University's Wolfson School - UK. He received his Ph.D. degree in Computer Science from PUCRS, Brazil in 2010. During his Ph.D., Dr. Ost worked as an invited researcher at the Microelectronic Systems Institute of the Technische Universitaet Darmstadt (from 2007 to 2008) and at the University of York (October 2009). After the completion of his doctorate, he worked as a research assistant (2 years) and then as an assistant professor (2 years) at the University of Montpellier II in France. He has authored more than 90 papers and his research is devoted to advancing hardware and software architectures to improve performance, security, and reliability of life-critical and multiprocessing embedded systems.

Ney L. V. Calazans holds a Ph.D. degree in Microelectronics from UCL-Belgium, obtained in 1993, and an M.Sc. in Computer Science and a BS in Electrical Engineering, both from UFRGS (Brazil), resp. obtained in 1985 and in 1988. He is a Professor at the PUCRS (Brazil) where he works since 1986. Since 1994 he is a permanent member of the CS graduate program (PPGCC) at PUCRS. During the 2014-2015 period he followed a Post-Doctorate Senior stage at the University of Southern California (USC) in Los Angeles, CA (USA). Prof. Calazans research interests include non-synchronous circuits, intrachip communication networks and EDA techniques and tools. He has authored around 200 publications on his fields of interest. He is a CNPq Researcher (PQ-1C), a Senior Member of the IEEE and a Member of the Brazilian Computer Society (SBC) and of the Brazilian Society of Microelectronics (SBMicro).



Rafael Garibotti (M'14–SM'22) is an Associate Professor at PUCRS University. Formerly he was a Visiting Scholar at Université Grenoble Alpes, France. He was also a Postdoctoral Fellow at both the prestigious School of Engineering and Applied Sciences of Harvard University, US and UFRGS, Brazil. He received his Ph.D. and MSc. Degree in Microelectronics, respectively from the University of Montpellier and EMSE, France and his BSc. Degree in Computer Engineering from PUCRS University. He is a distinguished Brazilian researcher (CNPq PQ-2 grant). His research activity focuses on AI safety, robotics and autonomous systems, multicore architectures and robust deep learning.

César Marcon (SM'19) is a Professor at PUCRS University, Brazil since 1995. He received his Ph.D. degree in Computer Science from the Federal University of Rio Grande do Sul in 2005. Professor Marcon is a Senior Member of the Institute of Electrical and Electronics Engineers (IEEE) and a member of the Association for Computing Machinery (ACM) and Brazilian Computer Society (SBC). He is a distinguished Brazilian researcher with a CNPq PQ-2 grant. He has more than 150 papers published in prestigious journals and conference proceedings. Since 2005, prof. Marcon coordinated several projects in health, telecommunications, and microelectronics areas with a total budget exceeding US$2 million.