Microprocessors and Microsystems 105 (2024) 105021

Contents lists available at ScienceDirect

Microprocessors and Microsystems

journal homepage: www.elsevier.com/locate/micpro

Non-intrusive study on FPGA of the SEU sensitivity on the COTS RISC-V VeeR EH1 soft processor from Western Digital

Daniel León [a,*], Juan Carlos Fabero [b], Juan A. Clemente [b]

[a] Higher Polytechnic School, Universidad Francisco de Vitoria, Spain
[b] Computer Architecture Department, Facultad de Informática, Universidad Complutense de Madrid (UCM), Spain

ARTICLE INFO

Keywords:
RISC-V
Fault injection
Soft error
COTS
VeeR EH1

ABSTRACT

This article studies the ISA-extension and application-specific soft error sensitivity of the RISC-V VeeR EH1 commercial processor core from Western Digital. To this end, a modified VeeRwolf SoC from Chips Alliance was deployed in a Digilent Nexys-A7 FPGA. Then, a fault injection platform was created for injecting soft errors in all architectural and micro-architectural registers of the VeeR EH1, without modifying the original processor core, when executing a set of commonly used space-related algorithms. Errors were categorized according to the consequences that they had on the normal execution of the processor, as well as to the unit of the core they were injected in. By changing compiling targets, four different combinations of RISC-V ISA extensions were also tested and compared, in the same processor IP, for a typical dot product algorithm, a hyperspectral imaging difference calculation and a SHA-256 hash. Experimental results will show how, for each one of these three case studies, the functionally equal binaries issued when compiling these programs using different ISA extensions are affected in different ways by error injections, opening the possibility to selectively compile functions based on a desired reliability/speed factor. The results additionally identify the specific units and subUnits within the processor's structure that have been affected, pinpointing the exact element where the bitflip occurred, after detecting an error.

1. Introduction and related work

RISC-V *Instruction Set Architecture (ISA)* is having an overwhelming acceptance, in particular in space applications, and it is called to be the ISA replacement for the current SPARC and PowerPC ISAs in Europe and in the USA respectively [1–3] due to its market neutrality, royalty-free model and good industry support. Early RISC-V rad-hard projects are aiming at much higher performance per watt and area efficiency [4], while still maintaining the low error rate in harsh environments. At the same time, a dynamic ecosystem of companies, tools and professionals contributes in the creation of even better opportunities for this ISA to thrive.

1.1. Space context and new players

Electronic components in space are exposed to ionizing radiation that may affect them temporarily (soft-errors) or permanently (hard-errors). As maintenance in space is usually not possible, different techniques have been applied to mitigate these effects, either by construction process, such as *Silicon-On-Insulator (SOI)* and triple well, or by circuit design, by adding redundancy at different levels, normally impacting on area and performance [5]. These radiation-hardened components are commonly referred to as *Radiation-Hardened By Process (RHBP)* and *Radiation Hardened By Design (RHBD)* respectively [6]. ISO standard 16290:2013 [7] defines the *Technology Readiness Level (TRL)* for space hardware, presenting nine different levels ranging from laboratory components (levels 1–4) to simulation-ready prototypes (levels 5, 6) and deployed components (levels 7–9). The qualification for obtaining TRL 8 *(pre-flight)* is usually long and expensive. Achieving TRL 9 (*flight-proven)* requires two years of deployment, therefore, qualifying components for space missions impacts heavily on time, cost, area and performance. For instance, the Mars 2020 Rover's brains are based on the BAE RAD 750 processor [8]. This rad-hard microprocessor, based on the PowerPC 750, was first launched in 2005 on NASA's Deep Impact, XSS-11, and Mars Reconnaissance Orbiter missions. Up to date, 28 RAD 750 processors have been launched [9] and, although BAE has a replacement processor ready, years-old technology is being used for space missions. On the other hand, *RISC-V High Performance Spaceflight Computer (HPSC)*, a 12-core RISC-V processor project for NASA, increases computing performance by a 1000-fold when compared with processors currently flying in space [3]. As space applications evolve,

* Corresponding author.
E-mail addresses: daniel.leon@ufv.es (D. León), jcfabero@ucm.es (J.C. Fabero), juanancl@ucm.es (J.A. Clemente).

https://doi.org/10.1016/j.micpro.2024.105021
Received 11 August 2023; Received in revised form 22 December 2023; Accepted 24 January 2024
Available online 1 February 2024
0141-9331/© 2024 The Author(s). Published by Elsevier B.V. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

more computing power is required and, therefore, qualification cycles must be shortened for certain non mission-critical processors, where soft errors could be mitigated, allowing for a small *Failure In Time (FIT)* number in exchange for a much bigger performance in less area and with a much smaller price tag. This trend is accelerated in an environment where private companies are entering the space industry. SpaceX, Facebook, Google or Amazon, among others, have moved the number of space electronics to a new level. As of January 2023, a total of 6718 satellites were orbiting Earth. SpaceX alone owns 3395 of them, a whooping 50.5%, considering all were launched in the past four years [10].

1.2. Previous research on RISC-V soft errors

Since the introduction of RISC-V, several studies have been conducted on the mitigation of the *Soft Error Rate (SER)*, using different platforms and techniques. *Triple Modular Redundancy (TMR)* and *Error Correcting Codes (ECC)* were applied to a Shakti processor targeting a 55 nm bulk *Application Specific Integrated Circuit (ASIC)*, offering fine-grain error detection [11]. TMR, alongside Hamming codes for protecting registers, is also used to harden the *Arithmetic Logic Unit (ALU)* and the control unit on a custom, low-cost RV32I processor testing CCSDS-123, a hyperspectral image compression algorithm running on a Xilinx Zynq ZC7020 [12]. Partial TMR is also applied to a Klessydra-T03 [13] multi-threaded soft processor, obtaining similar resilience to *Single Event Effects (SEEs)* when compared to full TMR in the same processor. Dual-Core LockStep (DCLS) is also applied as the main fault-tolerance technique in a custom-made RISC-V processor [14]. Several authors have used the *Soft Error Mitigation (SEM)* IP [15] from Xilinx for FPGA SRAM configuration memory protection, using *scrubbing* techniques, while testing soft errors on lowRISC processors [16], concluding that sensitivity is similar to other processors [17]. Other authors have focused on characterizing different modules and identifying critical bits in lowRISC [18] and Rocket [19] processors [20]. Some studies modify the processor at the *Register Transfer Level (RTL)* to test security on hidden registers on the Rocket processor [21] and obtain errors by module [22], concluding that there is a correlation between errors in different RISC-V processors that is not observed in other architectures. The UNSHADES1 project [23] inserts additional lines to the *clock enable* signals of the flip-flops and use partial reconfiguration on Xilinx Virtex FPGAs. This project has evolved into the FT-UNSHADES2 [24–26] offering a fast prediction of radiation effects in digital designs. Neutron radiation tests and TMR mitigation have also been explored into commercial FPGAs for the Taiga processor [27], the VexRiscv processor [28,29] and on the PolarFire FPGA [30], also used to expose a LowRISC soft processor to heavy ions [31].

Virtual Platforms (VP) and simulations may also be used for non-intrusive testing. Platforms like GEM5 [32] or QEMU [33] and frameworks such as SOFIA [34–36] explore the sensitivity against SEUs in a virtualized environment, at a much higher speed but without representing many microarchitecture features and memory behavior, therefore their accuracy in terms of performance and fault tolerance is very limited. In other words, the granularity of these platforms is considerably lower than that of emulation-based techniques. Due to these limitations, these virtual environments are usually recommended for testing during the first stages of design [6].

Finally, some authors have explored mitigation techniques based on software modifications on unmodified hardware. A modified C compiler, including extra instructions generating software redundancy, achieving a 2x–35.9x increase in *Medium Work To Failure (MWTF)* was used on a *Commercial Off-The-Shelf (COTS)* SiFive HiFive board [37] running matrix multiplication and SHA-256 algorithms, respectively, under neutrons [38]. Recently, Lodèa et al. [36] have shown, using SOFIA, that different compiler optimization options for a RISC-V ISA, thus producing different binaries, have an impact on the error rate and type when injecting soft errors, observing a 96% increase in program hangs when using the `-O2` optimization option as compared with using the `-O0` option.

1.3. Original contributions of this work

This work studies the SEE sensitivity of the COTS VeeR EH1 RISC-V processor by means of fault emulation on an FPGA. Although some works in the literature have addressed this problem before, mainly with academic RISC-V based architectures, the original contributions of this work with respect to the state of the art are listed below:

- The target core is a COTS device that is expected to have a broad deployment in the market in the future [39]. The analysis of experimental results study presented in this article is also a contribution since previous works on COTS processors focused on exposing an ASIC or commercial FPGA to radiation [30,38].
- This work studies the influence that including different combinations of ISA RISC-V extensions has in the SEE sensitivity of the VeeR EH1 processor, which to the best knowledge of the authors, has never been done so far. For this purpose, three space relevant applications will be evaluated for different extension combinations (RV32I, RV32IC, RV32IM and RV32IMC): a dot product function, a CCSDS-123.0-B-2 hyperspectral compression algorithm and a SHA-256 hash function, each of them having different workloads and features. This will make possible to identify which ISA-application pairs are especially resilient (or sensitive) against SEEs.
- The fault injection system, running on an FPGA emulation abstraction level, targets both architectural and microarchitectural (hidden) registers without affecting the RTL design nor changing processor timings, which, at this abstraction level, may prevent a commercial processor to cease working altogether. In this regard, related works either modify the processor core, run the fault injections at a different abstraction layer, or do not target all cpu registers (architecture and microarchitecture), therefore providing different data.

Table 1 presents a compilation of significant related articles and their main magnitudes.

2. Veer EH1 RISC-V (RV32IMC) processor from Western Digital

Western Digital is one of the founding members of the RISC-V International organization [40], created in 2015 to standardize and promote the RISC-V ISA and its software and hardware ecosystem. In 2019, Western Digital developed and open-sourced, through CHIPS Alliance [41], its VeeR EH1 Core (formerly known as SweRV EH1), based on the RV32IMC non-privileged specification. VeeR EH1 is a superscalar, 9-stage pipeline, in-order processor that achieves a 4.9–5.0 CoreMark/MHz score in an area range of $0.1\ mm^2$ at 28 nm TSMC [39,42]. EH1 is planned to be deployed in a billion storage controllers, providing an extensive test-field for its performance and reliability. Alongside this design, Western Digital has also open-sourced two other cores based on the EH1 design and the RV32IMC ISA: The VeeR EL2, a low-power implementation using a 4-stage pipeline with a performance of 3.6 Coremark/MHz; and the VeeR EH2, a dual-thread, 9-stage pipeline RV32IMAC implementation achieving 6.3 CoreMark/MHz and targeting 1.2 GHz at 16 nm.

VeeR EH1 RTL is available with no encryption, although the documentation on its microarchitecture is scarce. There is, however, commercial support available from Codasip. Additionally, Imagination Technologies launched in 2020 its *Imagination University Program (IUP) - RVFPGA* [43], providing training materials and laboratory workbooks for computer engineering students. These materials are built around the VeeR EH1 and providing some details on its microarchitecture. The VeeR EH1 1.8 core block structure is shown in Fig. 1 and uses 12,328 flip-flops, including those visible at the architecture level, such as the general purpose registers or the performance counters and the internal, microarchitecture-only elements. A breakdown of the core units and

Table 1
Related work comparison.

Author	Ref.	Year	Processor type	Abstraction	Injection method	Core modifications	Core registers tested	RISC-V ISAs	Injections	Number of binaries tested	Metrics
N. Gupta et al.	[11]	2015	Academic	RTL Simulation	XOR gates	Yes	Architecture and microarchitecure	1	Unknown	Synthetic set - Unknown	Percentage
Ramos et al.	[17]	2017	Academic	FPGA Emulation	FPGA CRAM	No	None	1	10,000	6	Percentage
Cho et al.	[22]	2018	Academic	FPGA Emulation	XOR gates	Yes	Architecture and microarchitecure	1	80,000	11	Percentage
Olivieri et al.	[13]	2019	Academic	RTL Simulation	TCL signal invert	No	Architecture and microarchitecure	1	Unknown	3	Error count
Mohseni et al.	[18]	2019	Academic	FPGA Emulation	FPGA CRAM	No	None	1	10,000	7	Percentage
Laurent et at.	[21]	2019	Academic	RTL Simulation	TCL signal invert	No	Architecture and microarchitecure	1	Unknown	1	Qualitative
Wilson et al.	[27]	2019	Academic	FPGA Radiation	FPGA Radiation	No	All SoC	1	Neutron	1	Error count
Dong et al.	[30]	2019	Commercial	FPGA Radiation	FPGA Radiation, Emulation	No	All SoC	1	Proton	1	Qualitative
Bandeira et al.	[35]	2019	Reference Definition	VP Simulation	Simulation	No	Architecture	1	800	1	Percentage
Santos et al.	[12]	2020	Academic	RTL Simulation	TCL signal invert	No	Architecture and microarchitecure	1	100	3	Error count
Aranda et al.	[20]	2020	Academic	FPGA Emulation	FPGA CRAM	No	None	1	27,000	Synthetic set - 4	Percentage
Oliveira et al.	[31]	2020	Academic	FPGA Radiation	FPGA Radiation	No	All SoC	1	Heavy ions	3	MWBF
Oliveira et al.	[31]	2020	Academic	FPGA Emulation	FPGA CRAM	No	None	1	Unknown	3	Percentage
James et al.	[38]	2020	Commercial	ASIC	ASIC Radiation	No	All SoC	1	Neutron	2	MWTF
Marques et al.	[14]	2021	Academic	FPGA Emulation	Software	No	Architecture	1	100,000	1	Percentage
Lodéa et al.	[36]	2022	Academic	VP Simulation	Simulation	No	Architecture	1	17,000	50	Percentage
This article		2023	Commercial	FPGA Emulation	Partial reconfiguration	No	Architecture and microarchitecure	4	38,800	10	Percentage

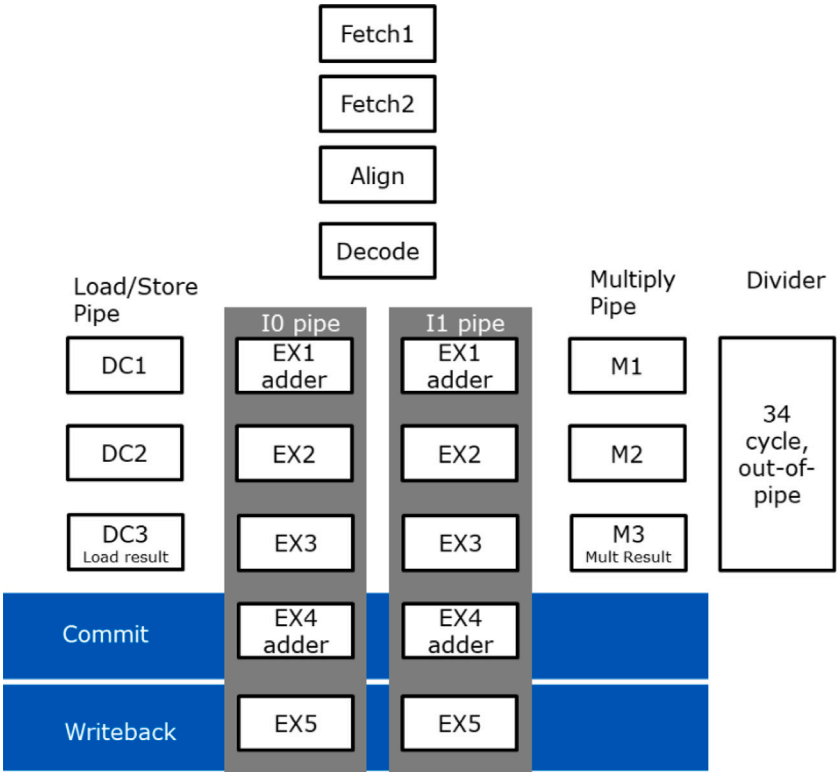


Fig. 1. VeeR EH1 block diagram [42].

subUnits, stating the number of flip-flops and a short description, is presented in Table 2.

ChipsAlliance offers an SoC [44] built around VeeR EH1 that adds a boot ROM, SDRAM support, an *Universal Asynchronous Receiver Transmitter (UART)* and a basic implementation of *General Purpose Input/Output (GPIO)* ports, running on a Digilent Nexys A7 (previously known as the Nexys 4 DDR). The research in this paper uses a modified version of this SoC, whose modifications are described in the following section, where the fault injection platform is explained in detail.

3. The fault injection platform

Our injection platform aims at testing effects for *Single Event Upsets (SEUs)* in both architectural and microarchitectural (i.e., not accessible

Table 2
VeeR EH1 1.8 : 12,328 Flip-Flop breakdown per unit/subUnit.

Unit	Flip-Flops	Description
DEC	**4165**	**Instruction decoder unit**
arf	992	Register file
decode	1227	Control signal generator
instbuff	544	Instruction buffers
tlu	1402	Commit/WB, flushing, exceptions
EXU	**2299**	**Execution unit**
alu	705	Early and late ALU
ap	144	ALU packet control
flush	256	Pipeline flushing
mul_div	323	Multiplier and divider
other_exu	123	Other EXU components
pp	292	Predict packet
src	456	Stage registers
IFU	**3279**	**Instruction fetch unit**
aln	855	Instruction alignment
bp	1506	Branch predictor
ifc	107	Fetch pipe control
mem_ctl	811	CCM and Cache control
LSU	**2362**	**Load & Store unit**
bus_intf	1135	Bus interface
clkdomain	6	Clock domain crossing
dccm_ctl	89	DCCM Control
lsu_i0_valid	5	Control for pipe
lsu_lsc_ctl	571	Load reserve/store control
lsu_single_ecc_err_dc	2	Error signaling
stbuf	554	Store buffer for DCCM
MEM	**117**	**Memory controller**
PIC	**106**	**Interrupt controller**

by the programmer) registers and flip-flops. To this end, we decided to test the unmodified VeeR EH1 core, as made available by Western Digital, running on an FPGA. The injection platform does not cover configuration bits within the FPGA as publicly available IPs, such as SEM IP from Xilinx, do a very good work in this regard. Note that characterizing FPGA configuration changes due to SEUs [17,20] is not in the scope for this research. However, previous works [43,45,46] have found that VeeR EH1 is very sensitive to time constraints, so changing the core itself to enable injections [22,23,47] would most likely lead to unexpected behavior or failures in the execution.

The procedure used to perform a bitflip is described below. This methodology was also used in a previous work made by the authors [48], although in that occasion, it was for the identification of bitflips occurred in FFs of the Nexys A7 FPGA:

1. The system clock is stopped (actually it is stretched).
2. A GCapture operation is issued to the FPGA via the JTAG interface. As a result, the state of every FF in the design is copied to the FPGA Configuration RAM (CRAM).
3. The frame `f` containing the FF where the injection is to be performed is read. The address of `f` and the offset of that FF within `f` is obtained from the *logic location* (`.ll`) file generated by the synthesis tools.
4. The bitflip is made on the bit of `f` that was selected in the previous step and the modified `f` is written back on the CRAM.
5. A GRestore operation is issued to initialize the values of all FFs to the ones indicated in the CRAM. As a result, all FFs will keep the same values that they had when GCapture was made (Step 2) except the FF that was modified in Step 4.
6. Finally, the system clock is restarted.

The proposed injection platform is based on an external controller, running on a *Raspberry Pi computer (RPi)*, connected to the modified VeeRwolf SoC running in the FPGA through its *Peripheral Module (Pmod)* connectors and the RPi GPIO ports. It comprises the blocks shown in Fig. 2:

- **FPGA synthesized hardware:**
  - **Chronometer and clock control (CCC):** Supporting cycle-accurate injections at any point during the execution, the processor is frozen when a specific cycle-count is reached.
  - **Injection Peripheral (INJP):** It is connected to the VeeR EH1 core, enabling the control of the chronometer and the transmission of information to an external control system running in a RPi, as also indicated in Fig. 2. The injection point request, chronometer start/stop and communication are all controlled from the software running in the VeeR EH1 processor, allowing relative injections from an execution point/subroutine within the tested program.
- **External control unit (RPi):**
  - **Injection Database:** A SQL database contains the bitstreams, the flip-flop list and the injection plan, organized into experiments and batch-runs, or campaigns, per experiment. It also records the experiments results.
  - **Control application and logging:** A *Graphical User Interface (GUI)* application is provided for managing and launching the experiment campaigns, controlling the actual bitflips using partial reconfiguration. Once launched, the experiment proceeds unattended, logging and recording the outcome of each injection.
  - **Data analysis and graphics:** Graphics, tables and alerts are automatically generated from the recorded data on each experiment, offering a comprehensive experiment cockpit.

The FPGA resources consumption for the fault injection system (i.e., CCC and INJP components) is very small, using roughly a 1% of the slices used by the VeeR EH1 processor. Table 3 shows a detailed breakdown of the number of slices, slice LUTs and slice registers available in the Artix A7 FPGA and those used in the SoC (without CCC and INJP) and EH1 processor, compared to the CCC and the INJP modules. Fig. 3 shows the detailed placement of the injection platform elements alongside the whole SoC containing the Veer EH1 processor.

To accommodate the injection platform, the VeeRwolf SoC needed to be modified. Main changes, with a brief explanation of the reason, are listed below.

1. **SRAM instead of SDRAM:** As the injection platform stretches the clock during the bit modifications in the FPGA, DRAM content and transfer operations may be affected. Static RAM does not suffer when the clock is altered.
2. **Clock generation:** VeeRwolf uses PLL for clock generation to the LiteDRAM Core, which operates at a higher frequency than the EH1 processor. LiteDRAM, then, generates the clock for the processor. As LiteDRAM is not used anymore, changes are done to the clock generation.
3. **Wishbone to AXI bridge modification:** The wishbone to AXI bridge in VeeRwolf presented some limitations in memory alignment for peripheral communication that were not suitable for the injection platform. These were corrected and new address spaces were created in the SoC *interconnect* peripheral for including the CCC and the INJP peripherals.

Fig. 4 presents a picture of the injection system, which offers some advantages over other available fault-injection solutions:

- Injects on unmodified CPU cores, not affecting timing constraints.
- May be used with encrypted IPs.
- Tests architectural and microarchitectural registers and flip-flops.
- Cycle accurate, with relative injection window set by the tested software.
- Small footprint in FPGA.
- Automatic injection campaign result cockpit and data analysis.

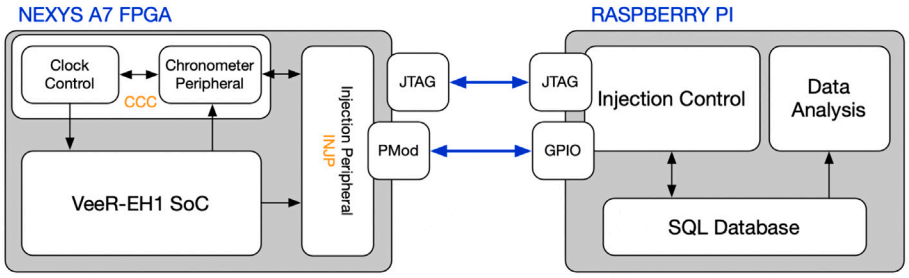


Fig. 2. Block diagram of the fault injection platform that was developed for these experiments.

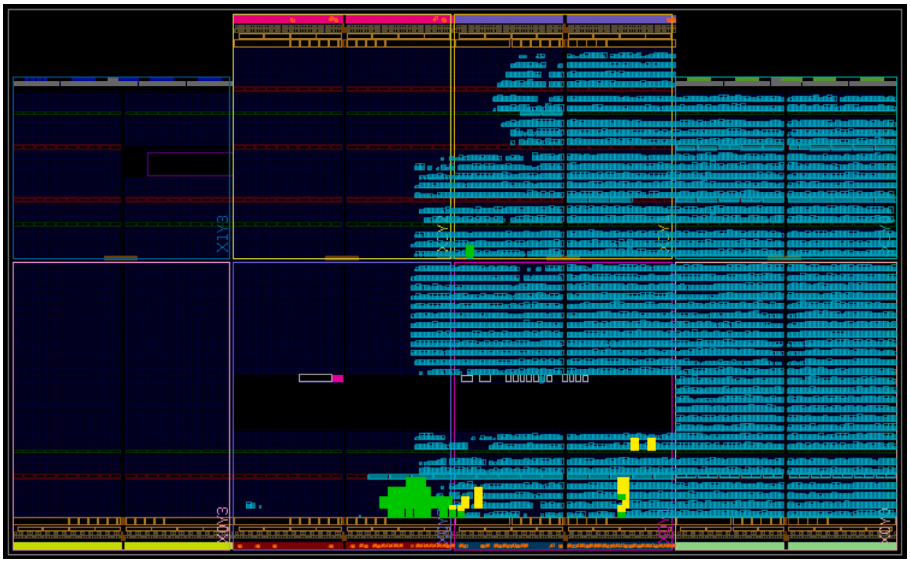


Fig. 3. Resources consumption of the modified VeeRwolf SoC on the Artix A7 XC7A100T FPGA, where the injection elements CCC and INJP are displayed in green and yellow, respectively. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Table 3
FPGA utilization of the injection system components (INJP and CCC), compared to the Veer EH1 processor, the whole VeeRwolf SoC and the resources available in the Artix A7 XC7A100T FPGA.

Entity	Slices	Slice LUTs	Slice registers
Artix A7 XC7A100T	15,850	63,400	126,800
VeeRwolf SoC	8031	28,151	13,878
Veer EH1	7545	26,366	12,328
CCC	70	114	245
INJP	15	13	42

- Built using inexpensive and readily available components.

The proposed injection system also presents some limitations when compared to other approaches. Running at about 0.2 s per injection in a single test-board environment (fully parallel execution is possible by adding more boards), it is slower than non fine-grain software simulators, such as GEM5 [32], virtual platforms like SOFIA [34] and, potentially, RTL-modified based testing [22] on FPGA. However, equivalent state-of-the-art fine-grain software RTL simulators run at as low as 1 KHz on modern CPUs [49,50] making simulation at this granularity much slower than the proposed one.

Fig. 4. Photograph of the fault injection platform.

4. Experiment conditions

For this experiment we used a set of C-language algorithms common in space applications: dot product (DOT) (16 × 16 16-bit true-random generated numbers), the hyperspectral algorithm CCSDS-123.0-B-2 [51] central difference calculation (CCSDS) for a single pixel of an test image containing 36 × 100 pixels and 17 spectral bands, and a SHA-256 hash (SHA) using a sample string defined by the user; in this case, “RISC-V”. These baremetal source codes are then compiled using four different

Table 4
Details on binaries for different algorithm-ISA target combinations and the calculated number of injections (n) required in a population (N) for an error of 1.0% with a confidence level of 95%. Cycles = execution cycles.

ISA	DOT			CCSDS			SHA		
	cycles	N	n	cycles	N	n	cycles	N	n
RV32I	2241	27,627,048	9600	1859	22,917,752	9599	7745	95,480,360	9603
RV32IC	2156	26,579,168	9600	1856	22,880,768	9599	7732	95,320,096	9603
RV32IM	708	8,728,224	9593	380	4,684,640	9584	–	–	–
RV32IMC	724	8,925,472	9593	362	4,462,736	9583	–	–	–

ISA targets: RV32I, RV32IC, RV32IM and RV32IMC respectively, always considering that the VeeR EH1 remains unchanged, so certain data and control paths remained unused with some compilation targets. The RV32I ISA is one of the frozen base ISAs for RISC-V [52], containing 40 unique instructions for a 32-bit integer ALU. The standard extensions for *Compressed Instructions (C)* and *Multiplication and Division (M)* are also frozen and may be added to the base ISA to support 16-bit instructions and integer multiplication and division respectively. Note that the C compiler never used the M extension for the SHA algorithm, therefore a compilation to the RV32I and RV32IM targets will produce the same binaries. The same thing happens with RV32IC and RV32IMC. For this reason, the analyses for M-enabled targets on SHA have been omitted.

Details are shown on Table 4, showing *Population (N)*, the number of potential candidates for fault injection, obtained by multiplying the number of flip-flops in the Veer EH1 core by the execution cycles and *n*, the number of required injections for a confidence level of 95% and an error margin of 1.0% as defined by the calculation presented by Leveugle et al. [53].

$$n = \frac{N}{1 + \mathrm{err}^2 \times \frac{N-1}{t^2 \times p \times (1-p)}} \tag{1}$$

The **p** parameter corresponds to the expected percentage of errors resulting in a failure, and, as proposed by the author, remains fixed at 0.5. This sample gives a correct estimation on whether the program would succeed or fail in its execution. Afterwards, the type of failure is identified, as well as the conditional probability of each type of failure given the unit where the injection occurred.

However, an important condition for this experiment is ensuring that all four ISAs are exposed to exactly the same injections, which occur at a specific cycle in a specific flip-flop, as the binaries being executed are different with each target ISA compilation. The experiments expose the different binaries to the same conditions. Therefore the fault injection campaign is cloned for each of the ISAs in all three tested algorithms. The selected cycle model for fault injection follows a uniform distribution and, since the experiment replicates the same injections for all ISAs and the execution speeds of the four binaries may be different, some of them may occur after the program is finished. As an example for the DOT, in order to make sure that the number of fault injections carried out over 708 cycles (which is the execution time of the fastest ISA - RV32IM) is significant enough (thus within the target error and confidence level), the number of injections over 2241 cycles (slowest ISA execution cycles - RV32I) has to be adjusted to 30,365. This corresponds to a roughly 0.49% error for the longer experiments, while keeping the error margin below 1.0% for the shortest. The final number of injections was rounded up to 40,000 per experiment for this specific algorithm. Fig. 5 is a representation, using the dot algorithm as an example, of the actual number of fault injections that were performed in each clock cycle, randomly carried out over 2241 clock cycles, which is the execution time of this algorithm using the RV32I ISA target. This is the slowest one among the four ISAs that were tested for this algorithm (see Table 4).

Each of the injections is performed on a clean run of the tested programs, recording the effect that each single event may have had in the expected outcome of the experiment. Since the total number of fault injections is way higher than the number of execution cycles of the program, this figure shows the number of times a fault was injected (randomly in a FF of the VeeR EH1 core), in each one of the 2241 execution cycles of the DOT algorithm.

Table 5 shows the total number of fault injections performed in each algorithm-ISA combination and the calculated statistical error. The number of said injections (40,000 for DOT, 51,000 for CCDS and 10,000 for SHA) and injection points (cycle, FF) is fixed across all ISAs for the same algorithm. However, as mentioned above, the algorithms' execution times decrease as the C, M and MC extensions are added, leading to less effective injections, for a given algorithm, in ISAs with shorter execution times, as shown in the table.

We have used a modified injection classification framework as presented by Weaver et al. and Shubu Mukherjee [5]. Each of the fault injections is classified according to the following criteria:

- **NO EFFECT:** The fault injection had no effect on the expected outcome of the program. This case is further classified attending to whether the injection has occurred before or after the program was finished. In the latter case, all fault injections fall into this category as the SEU has no effect on an already completed execution of the program. This is recorded to compare different-speed campaigns over a fixed amount of time.
- **ERROR**: The injection has caused the program to not end as expected. The injection is then categorized as follows:
  - **SED:** *(Single Event Delay)* - The injection has caused the program to take more cycles (up to double the time) than expected to finish. However, the result is correct. These errors will be represented in green in Figs. 9, 11 and 13.
  - **DUE:** *(Detected Unrecoverable Error)* - The injection has prevented the program to finish in, at least, twice the time it was expected to. Since the experiment runs baremetal programs, the actual effect is a system halt. These errors will be represented in yellow in Figs. 9, 11 and 13.
  - **SDC:** *(Silent Data Corruption)* - The program has finished in time, but the calculation outcome is wrong. These errors will be represented in red in Figs. 9, 11 and 13.

Depending on the application, not all errors are equally harmful. While real-time systems require to comply with timing constraints, some applications would not be severely affected by a longer execution time. If this is the case, SED errors may be acceptable and no further action would be required. In other cases, a system watchdog or process in the operating system could restart the task if it stops responding or crashes, so, while DUE errors are more severe than SED errors, a system could still detect and, sometimes, even recover from DUE errors. SDC errors, however, could only be detected if the system knows the result in advance or, at least, the acceptable range of results. In any case, there would certainly be cases where the error remains undetected and passed over to additional processing stages, potentially increasing the damage in the correctness of the data. Hence the color coding selected for these errors in the figures presented in the next section.

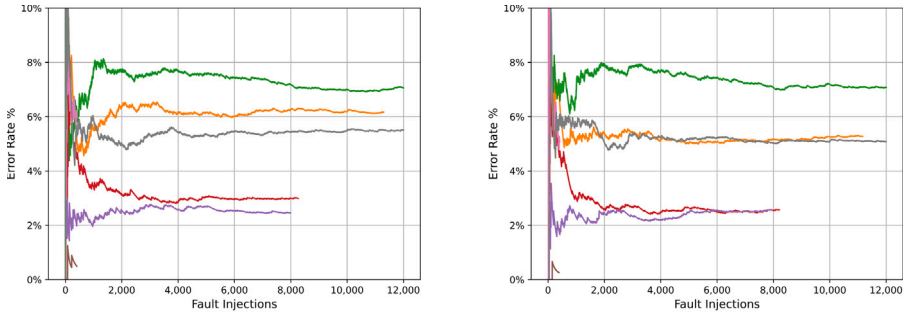
5. Experimental results

A total of ten fault-injection campaigns were performed, one per each algorithm and ISA target combination, considering that the SHA

Fig. 5. Injections per cycle (blue) and average injections per cycle (orange) for the DOT algorithm. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Table 5
Total number of fault injections and calculated statistical error for each algorithm and combination of ISA extensions evaluated.

ISA	DOT		CCDS		SHA	
	Injections	Sta. Error	Injections	Sta. Error	Injections	Sta. Error
RV32I	40,000	0.49%	51,000	0.43%	10,000	0.98%
RV32IC	38,736	0.50%	50,910	0.43%	9979	0.98%
RV32IM	12,586	0.87%	10,436	0.96%	–	–
RV32IMC	12,748	0.87%	9942	0.98%	–	–



(a) Target RV32I (b) Target RV32IC

(c) Target RV32IM (d) Target RV32IMC

Fig. 6. Accumulated error rate (%) per unit and fault injection step (4 ISAs), for the DOT algorithm. Note that the number of injections for the "Full CPU Core" lines correspond to the aggregated addition of the remaining ones (i.e., the CPU units). For the sake of clarity, the plots have been zoomed out to the first 12,000 fault injections, as said lines do not yield significantly different results beyond this threshold.

algorithm is only analyzed for two ISAs, as the compiler does not make use of the M extension in this case.

Fig. 6 shows the evolution of the error rate (%) of the whole core ("Full CPU Core") and per unit (IFU, DEC, LSU, EXU, PIC and MEM), for the DOT algorithm and the 4 ISAs considered (RV32I, RV32IC, RV32IM and RV32IMC). Displayed values are stable beyond a few thousands of fault injections, which confirms that the number of performed injections (40,000 for the DOT, see Table 5) yield consistent and reliable results.

Fig. 7 presents the fault injection error rates of different types (SEDs, DUEs and SDCs) for all algorithms and ISAs. Next, Fig. 8 offers a more detailed view of the results, using a different arrangement in order to classify the observed errors in the DEC, EXU, IFU and LSU Units, per ISA and algorithm (DOT, CCSDS and SHA). MEM and PIC units have been left out of this figure for space reasons. Since their flip-flop count is much smaller than the other units, their error rate is not significant and will not even be shown in the figures.

Looking at Figs. 7 and 8, one can observe that, when an SDC error is detected, the originating injection is more likely to be in the DEC unit and, secondly in the EXU, across all algorithms and ISAs. Similarly, DUE errors have, primarily, originating injections in the DEC unit, followed

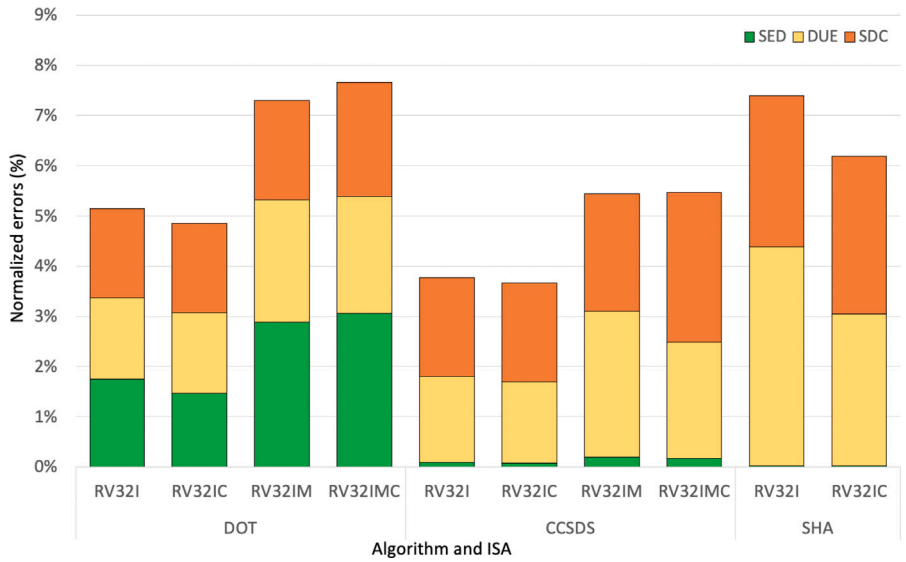


Fig. 7. Error rates (%) on the RISC-V VeeR EH1 core, categorized by type (SED, DUE or SDC) per algorithm and ISA.

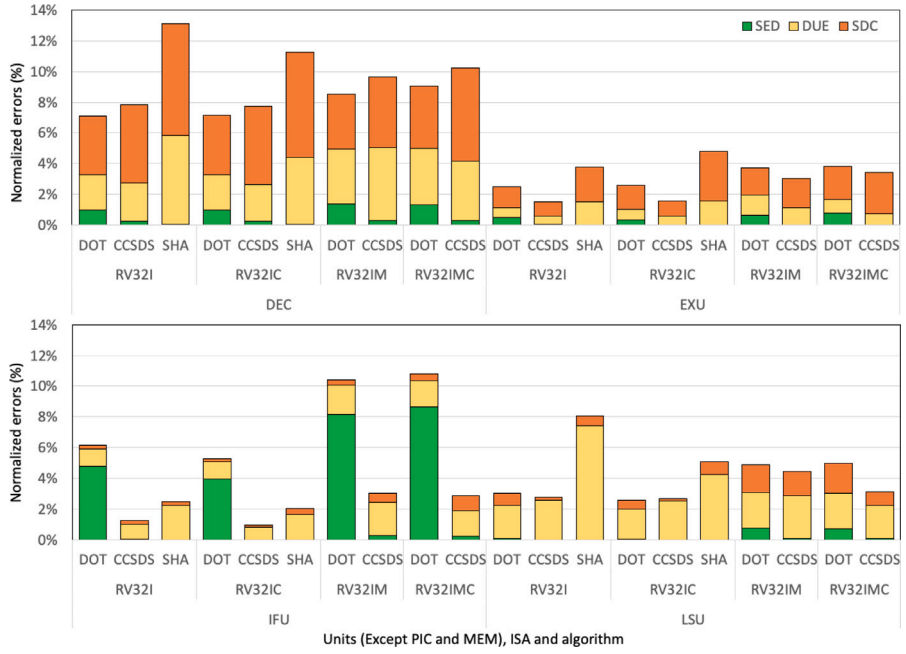


Fig. 8. Error rates (%) on the RISC-V VeeR EH1 core, categorized by type (SED, DUE or SDC) per relevant unit (except PIC and MEM), ISA and algorithm.

by the LSU. Lastly, SED errors appear primarily in the DOT algorithm and, once observed, the originating unit is, almost always, the IFU.

Table 6 shows the calculation for the *Architecture Vulnerability Factor (AVF)*, introduced by Mukherjee [54], which is used to estimate the failure rate of a processor by using a probability of a fault resulting in an error in the outcome of the program. This can be estimated by performing fault injections or by using an *Architecturally Correct Execution (ACE)* analysis, although the latter technique may overestimate the AVF in up to 3.5x [55]. These calculations only consider the correctness of the result, not the potential delays, therefore only SDC and DUE errors are accounted for. The injection campaigns result in an overall AVF of 4.0%. As detailed further in the document, most of the observed errors originate in the decoder unit across all ISAs. Also, using an RV32IC ISA leads towards the lowest AVF, whereas using the RV32IMC results in the highest AVF.

Table 6
AVF calculations considering SDC and DUE errors.

Unit	ISA				Total
	RV32I	RV32IC	RV32IM	RV32IMC	
DEC	7.5%	7.3%	7.9%	8.5%	**7.6%**
EXU	2.0%	2.2%	3.1%	3.2%	**2.3%**
IFU	1.4%	1.3%	2.4%	2.3%	**1.5%**
LSU	3.4%	2.9%	4.2%	3.8%	**3.3%**
MEM	1.4%	0.7%	1.5%	1.1%	**1.1%**
PIC	1.2%	0.3%	0.6%	0.0%	**0.7%**
Veer EH1	**4.0%**	**3.8%**	**4.7%**	**4.8%**	**4.0%**

In the next subsections, two sets of graphical results are presented per each algorithm. The first set shows how the fault injections, affecting the flip-flops in the core units, generate different error types as the

Fig. 9. DOT soft errors per CPU unit and relevant subUnit over the full execution time for all four ISAs.

injections occur in a specific cycle and flip-flop. The relevant subUnits of the CPU are also included. The second set present the conditional probability of an error source being an injection in an specific unit, once the error has been observed. In all cases, different units and subUnits are susceptible to be affected in specific ways by the fault injections.

5.1. DOT results

This section contains the particular results for the DOT algorithm, compiled with the four available ISAs. Fig. 9 shows the types of errors that each fault injection produced in the core, for the 4 campaigns that were carried out; featuring RV32I, RV32IC, RV32IM and RV32IMC ISAs. X axis represents the clock cycle when the bitflip was injected, whereas the Y axis represents the target FF. In the latter case, FFs are categorized by unit/subUnit containing each one of the FFs in the VeeR EH1 microarchitecture. As mentioned above, the clock cycle and the FF to inject to were randomly selected each time. This description also holds for Figs. 11 and 13 (which will be shown in Sections 5.2 and 5.3, respectively).

Then, Fig. 10 presents, for the four ISAs, the conditional probability of an error being injected in a particular unit, once an error type has been observed. This description also holds for Figs. 12 and 14 (which will be shown in Sections 5.2 and 5.3, respectively).

Looking at both figures, one can roughly observe that, for this algorithm, the DEC unit is the most affected one by SDCs and the IFU, by SEDs. The rest of the units experience all types of errors, although with less density.

5.2. CCSDS results

This section contains the particular results for the CCSDS algorithm, compiled with the four available ISAs. Similarly to the previous algorithm, this is shown in Fig. 11 and in Fig. 12.

This time, most common errors are SDCs and DUEs, with very little presence of SEDs. SDCs are more present in the DEC and the EXU-*alu*, whereas DUEs are scattered around the rest of the units.

5.3. SHA results

This section contains the particular results for the SHA algorithm, compiled with the two available ISAs: RV32I and RV32IC, as the compiler never make use of the M extension for this specific algorithm. Results are presented in Figs. 13 and 14.

Similarly as Fig. 11, the DEC and EXU-*alu* units are most affected by SDCs. Here, it is also interesting to note that the SHA accumulates most of the SDCs in the second half of its execution (Fig. 13), which is when the algorithm actually makes the hashing operations.

6. Discussion

This section presents an in-depth interpretation of the results presented in Section 5. To this end, a profiling was made, using the RISC-V ISA simulator Spike [56], in order to characterize the workload of the codes of all three algorithms, classifying the instructions that were executed according to their types: “Load/Store”, “Branches”, “ALU” and “Others”, all in both their “Standard” and “Compressed” (C-extension) form and, finally, the “Multiplication” (M-extension). This classification is presented in Table 7.

6.1. Workload dependence

The first observation that can be made from Fig. 7 is that the type of observed errors depends on the workload of the program. Thus, firstly, SEDs are considerably more common in the DOT algorithm than in the other two. These kind of errors consist in the result being correctly calculated later than expected, which might be caused by an error affecting the branch prediction system of the processor or alteration of the I-cache control (IFU-*aln*). This is consistent with the data displayed in Fig. 8, which shows that the IFU (containing the core’s branch predictors and instruction alignment logic) is the unit that concentrates the majority of SEDs, especially for the DOT, being the simplest algorithm, followed by the CCSDS. SEDs also occur in the

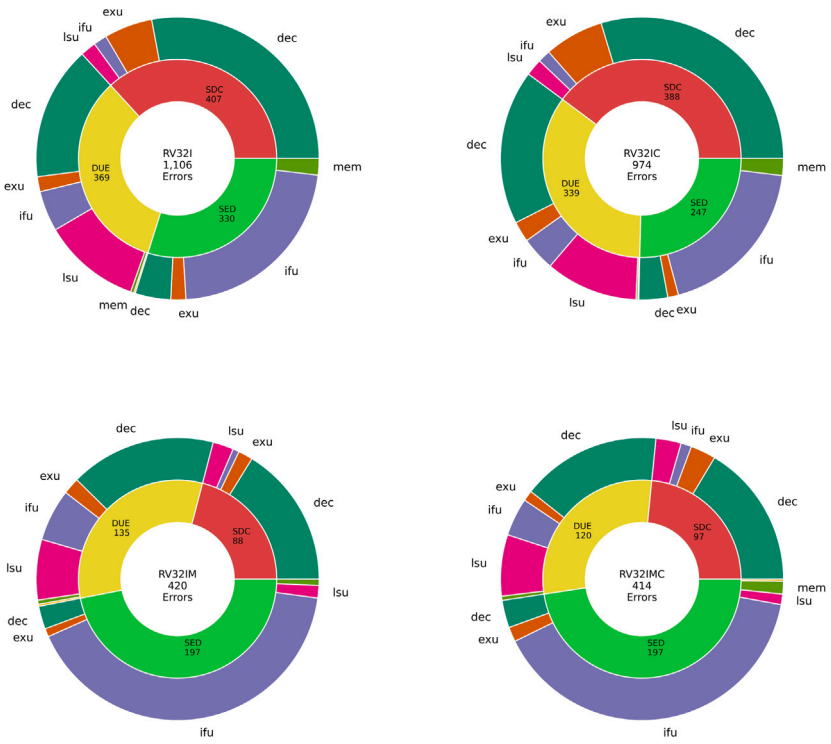


Fig. 10. DOT - Conditional probability of originating bitflip unit given an error category (4 ISAs).

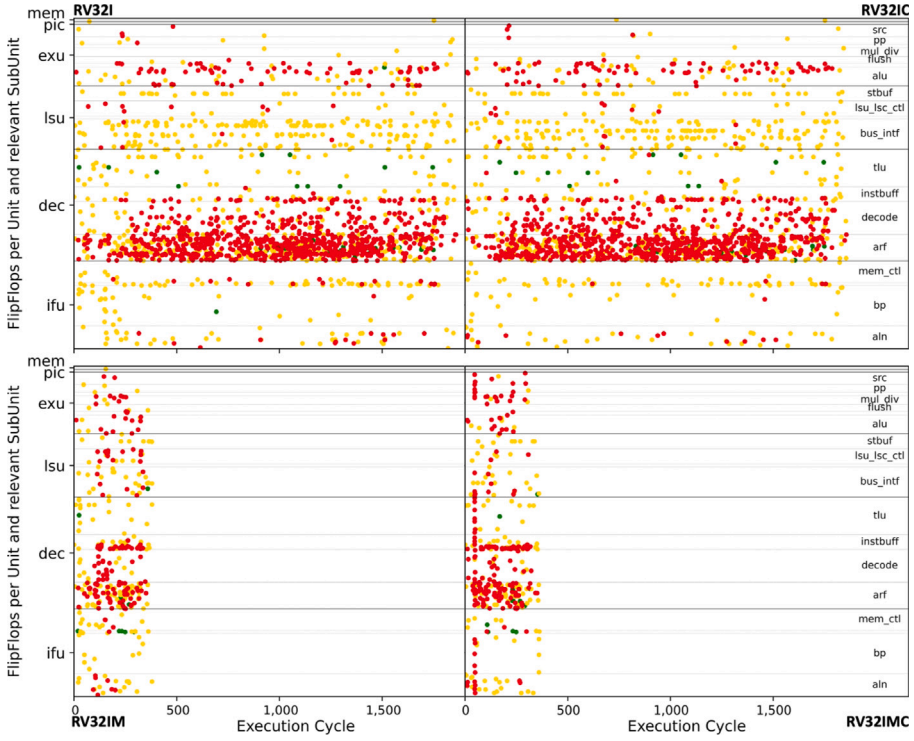


Fig. 11. CCSDS soft errors per CPU unit and relevant subUnit over the full execution time for all four ISAs.

DEC, EXU and LSU units (again, most probably for the DOT), but much less frequently than in the IFU.

Secondly, there exists a significant difference in SEDs between the DOT and CCSDS, which cannot simply be explained by their instruction profiling, since it was similar in both algorithms (~15%–18% Load/Store, ~15% Branches and ~55% ALU for RV32I), these percentages being similar between both algorithms, and across all ISAs. Although this profiling would explain why DUE and SDC errors are very similar for the DOT and the CCSDS, but not the difference observed in SEDs in a direct manner. However, the program flow in CCSDS is less

Fig. 12. CCSDS - Conditional probability of originating bitflip unit given an error category (4 ISAs).

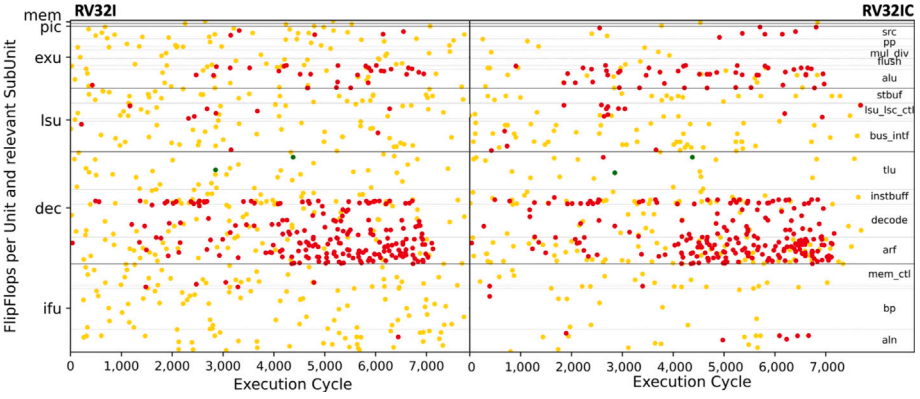


Fig. 13. SHA soft errors per CPU unit and relevant subUnit over the full execution time for RV32I and RV32IC.

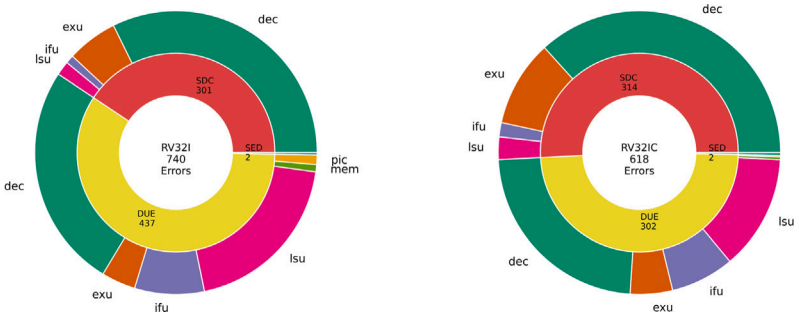


Fig. 14. SHA - Conditional probability of originating bitflip unit given an error category (2 ISAs).

Table 7
Instruction profiling per algorithm, ISA and instruction type. (number of instructions & percentage of the total executed instructions).

Algorithm	ISA	Load/Store standard	Load/Store compressed	Branches standard	Branches compressed	ALU standard	ALU compressed	Others standard	Others compressed	Multiplication	Total
DOT	RV32I	228 (14.99%)	0 (0.00%)	428 (28.14%)	0 (0.00%)	862 (56.67%)	0 (0.00%)	3 (0.20%)	0 (0.00%)	0 (0.00%)	1521
	RV32IC	187 (12.29%)	41 (2.70%)	410 (26.96%)	18 (1.18%)	694 (45.63%)	168 (11.05%)	2 (0.13%)	1 (0.07%)	0 (0.00%)	1521
	RV32IM	210 (59.15%)	0 (0.00%)	20 (5.63%)	0 (0.00%)	106 (29.86%)	0 (0.00%)	3 (0.85%)	0 (0.00%)	16 (4.51%)	355
	RV32IMC	171 (48.17%)	39 (10.99%)	18 (5.07%)	2 (0.56%)	2 (0.56%)	104 (29.30%)	2 (0.56%)	1 (0.28%)	16 (4.51%)	355
CCSDS	RV32I	158 (18.31%)	0 (0.00%)	234 (27.11%)	0 (0.00%)	465 (53.88%)	0 (0.00%)	6 (0.70%)	0 (0.00%)	0 (0.00%)	863
	RV32IC	103 (11.94%)	55 (6.37%)	212 (24.57%)	22 (2.55%)	367 (42.53%)	98 (11.36%)	2 (0.23%)	4 (0.46%)	0 (0.00%)	863
	RV32IM	134 (59.29%)	0 (0.00%)	12 (5.31%)	0 (0.00%)	56 (24.78%)	0 (0.00%)	6 (2.65%)	0 (0.00%)	18 (7.96%)	226
	RV32IMC	85 (37.61%)	49 (21.68%)	8 (3.54%)	4 (1.77%)	12 (5.31%)	44 (19.47%)	2 (0.88%)	4 (1.77%)	18 (7.96%)	226
SHA	RV32I	5168 (44.14%)	0 (0.00%)	224 (1.91%)	0 (0.00%)	6306 (53.86%)	0 (0.00%)	10 (0.09%)	0 (0.00%)	0 (0.00%)	11,708
	RV32IC	4831 (41.24%)	340 (2.90%)	212 (1.81%)	13 (0.11%)	1732 (14.79%)	4576 (39.06%)	2 (0.02%)	8 (0.07%)	0 (0.00%)	11,714

prone to achieve performance gains of I-cache (temporal and spatial locality) than a more regular and loop-based DOT algorithm and the entropy of the SHA algorithm is greater, so branch prediction is not as straight-forward as it is with DOT, so alterations on the branch prediction subUnit would probably have less impact.

Next, DUE errors are more common in the SHA than in the other two algorithms. The profiling in Table 7 indicates that this algorithm is considerably the most memory-bound one (~44% Load/Store vs. ~15%–18% for the other two). This is consistent with data shown in Fig. 8, which indicates that DUEs significantly increase in the LSU for the SHA, both in the RV32I and RV32IC targets. This phenomenon might be caused by incorrect memory accesses, or significant I-cache misses caused in this unit that delays the execution of the program leading to DUE timeouts. Slight increments in DUEs and SDCs are also observed in the DEC, EXU and IFU units, when comparing the SHA with the other two algorithms. Again this could be due to the longer execution time, higher resource usage and a much more complex call graph for this algorithm compared to the other two.

Finally, SDCs are again more common in the SHA than in the other two applications. The ALU instructions (6306, see Table 7) are more abundant in this algorithm when compared to the other two (862 and 465), and so are the number of registers used in the register file, the subUnit responsible for most of these errors, which would explain this behavior.

6.2. Unit and subunit dependence

Fig. 8 indicates that the DEC unit is, by far, the one that is most affected by all types of errors (especially SDCs), for all the targets and algorithms. This unit contains the register file of the core, whose bitflips significantly affect the normal operation in a 3-address architecture core, where all operations must be performed with data in registers. Looking by subUnits (Figs. 9, 11 and 13), the *decode*, *arf* and *instbuff* (described in Table 2) are the most affected ones by SDCs.

SDCs are very significant in the EXU unit too, which are due to malfunctions in the ALUs of the core. DUEs are also significant in this unit since these ALUs are also used for calculating branch target addresses. In these cases, a wrong target address in a branch would make the program execute a wrong piece of code, thus leading to a system halt.

In Figs. 9 and 11, it is also interesting to note that the *mul_div* subUnit is less populated with errors (of all types) for targets RV32I and RV32IC than when using the M extension (RV32IM and RV32IMC). This makes sense since this subUnit is used by multiplication instructions that can only exist in the code when the M ISA extension is present. However, a few SED and DUE errors can still be observed in this subUnit for RV32I and RV32IC (again, both in Figs. 9 and 11), which shows that errors affecting a module of the processor (in this case, the multiplication unit) that is theoretically not used by a program can also affect the correct execution of that program.

Most of the errors affecting the LSU unit were DUEs, with a few SDCs and some SEDs, the latter mainly for the DOT. These SDCs and DUEs might be caused by exceptions raised by wrong memory accesses, or fetching instructions from wrong memory positions. Fig. 8 reveals that the *lsu_lsc_ctl* subUnit concentrates most of the SDCs of this unit, whereas *stbuf* and *bus_intf* are more affected by DUEs.

Finally, analyzing the units being affected by the different types of errors (Figs. 10, 12 and 14), it seems clear that the majority of SDCs occurred in the DEC unit, whereas the DUE errors affected mostly the LSU and DEC units, followed by the IFU. Concerning the SED errors, they mostly affect the IFU unit (Fig. 10), but only for applications that can benefit from branch prediction (as seen in Section 6.1 for the DOT algorithm, Fig. 9). Otherwise, there is a residual amount of SEDs mostly affecting the DEC unit (Figs. 12 and 14).

6.3. ISA dependence

Looking at Fig. 7, it can be observed that the error rates of ISAs including the M extension are higher than those that do not use this extension. This holds when comparing RV32I vs. RV32IM and RV32IC vs. RV32IMC, for the DOT and CCDS. On the one hand, using target M makes the application being more memory bound than not using it (see Table 7), which explains the increase observed of DUEs in all cases. On the other hand, more HW resources (for decoding and executing multiplications) are used when this target is activated, therefore making the core more prone to SDC errors. This phenomenon can also be observed in Figs. 9, 11 and 13, where some subUnits (especially *DEC-instbuff*, *IFU-aln* and *EXU-muldiv*) have greater concentration of errors when using the M extension.

The effects of using the M extension are also visible in the pie charts of Figs. 10 and 12. Thus, for the DOT, the relative abundance of SEDs increases when using the M extension (RV32I vs. RV32IM and RV32IC vs. RV32IMC), and so does the conditioned probability that these SEDs have occurred in the IFU. For the CCSDS (Fig. 12), the relative abundance of DUEs increases instead. Both results have in common a decrease of the relative abundance of SDCs, which are the most critical errors. This is consistent with the fact that including M makes the programs being more memory-bound, in a similar way as DUEs were more common in the SHA than in the other two algorithms (this was discussed at the end of Section 6.1).

Finally, the usage of the C extension in absence of the M extension seems to slightly decrease the error rates as a whole (Fig. 8). However, the effect is the opposite if the M extension is included in the target (i.e., RV32IMC vs. RV32IM). The profiling of Table 7 reveals that, for the RV32IC target, the percentage of "ALU Compressed" instructions (20.5% on average) is much higher than the "Load/Store compressed" ones (4% on average). However, for RV32IMC, these percentages are 24.4% and 16.3%, respectively. Therefore, it looks like that an intensive use of "ALU Compressed" instructions increase the reliability of the core against soft errors, whereas the "Load/Store Compressed" ones have the opposite effect. This could be related to the extra resource usage of the RISC-V compressed load and store instructions, where the stack pointer plays an important role in calculating the effective memory address and some operations are needed in the process, while the ALU instructions do not use any extra registers nor require additional calculations. Also, the relative abundance of SDCs seems to increase

when the C extension is used, but only when the M extension is also in the target ISA (therefore, the application is more memory-bound, see RV32IM vs. RV32IMC in Figs. 10 and 12). The same phenomenon is observed in Fig. 14 for the SHA application.

7. Conclusions and future work

This article has presented an analysis of experimental results study of the soft-error sensitivity of the COTS RISC-V VeeR EH1 processor by using fault injection. Different combinations of ISA RISC-V extensions (RV32I, RV32IC, RV32IM and RV32IMC) were evaluated when executing three space-relevant applications: a dot product function, a CCSDS-123.0-B-2 hyperspectral compression algorithm and a SHA-256 hash function, each of them having different workloads and features. A profiling on each one of these applications was firstly made using the RISC-V Spike simulator and then, faults were injected during the execution of these benchmarks, randomly both during their execution and on any microarchitectural FF existing in the core. The fault injection system that was developed is non intrusive and autonomous.

Fault-injection campaigns were performed with statistically significant amounts of bitflips that allowed gathering experimental results with a confidence level of 95% and an error margin of 1%. The following conclusions can be drawn:

- The types and severity of errors observed in the core had a strong dependence with the unit where they occurred. Thus, once SDC/DUE errors were observed, the highest probable originating unit is, by far, the DEC unit. Therefore, if only one unit can be hardened against radiation, it should be this one.
- There exists a strong correlation between the different types of errors and the units/subUnits mostly affected by them. Thus, SDCs mostly affect the DEC unit, DUEs mostly affect LSU, DEC and IFU, and SEDs (mainly present in loop-based programs) mostly affect the IFU.
- There exists a strong correlation between the target ISA extensions and errors affecting certain elements of the core. Thus, more errors were observed in the multiplier when the M extension was present in the target ISA.
- Including the M extension increases the error rates of all types, and especially for those that are less critical (SEDs, for the DOT application and DUEs, for the CCSDS). However, execution times are significantly lower. A speed/reliability assessment may be needed when deciding target ISA for specific applications.
- Memory-bound programs (such as the SHA) have highest SDC and DUE error rates. This effect is also observed when using the M extension in the other algorithms.
- Loop-based programs (such as the DOT) have higher SED error rates. However, the specific error rate depends on the specific application that is targeted.
- The usage of the C extension leads to interesting results: if compilation leads to more "Load/Store Compressed" instructions, then all error rates increase. Thus, this was observed for the RV32IMC ISA. However, for RV32IC, error rates decrease instead, the reason being that more "ALU Compressed" instructions are generated after compilation in the absence of the M-extension for the target ISA.

Future work will involve fault injection on the complete CCSDS 123.0-B-2 hyperspectral compression algorithm [51,57] (as only a small routine was considered in this work) and studying the F (Floating Point) ISA extension [58–60]. Studying the (Zbk, Zvk*) cryptography extensions [61], such as the ones proposed by Fritzmann [62] and Nannipieri [63] in an scenario of post-quantum cryptography (PQC) [64–67] and complementing these results with radiation-ground tests under protons or neutrons are also interesting lines for future research.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Juan Antonio Clemente reports financial support was provided by SPANISH MICINN project PID2020- 112916GB-I00.

Data availability

Data will be made available on request.

Acknowledgment

This work was supported by the Spanish MICINN project PID2020-112916GB-I00.

References

[1] S. Di Mascio, A. Menicucci, G. Furano, C. Monteleone, M. Ottavi, The case for RISC-V in space, 2019, pp. 319–325, http://dx.doi.org/10.1007/978-3-030-11973-7_37.
[2] R. Weigand, A. Fernández, RISC-V in space, http://microelectronics.esa.int/riscv/rvws2022/index.php.
[3] S. Leibson, NASA Recruits Microchip, SiFive, and RISC-V to Develop 12-Core Processor SoC for Autonomous Space Missions, URL https://tinyurl.com/5h7b5s34.
[4] N.-J. Wessman, F. Malatesta, S. Ribes, J. Andersson, A. García-Vilanova, M. Masmano, V. Nicolau, P. Gomez, J.L. Rhun, S. Alcaide, G. Cabo, F. Bas, P. Benedicte, F. Mazzocchetti, J. Abella, De-RISC: A complete RISC-V based space-grade platform, in: 2022 Design, Automation & Test in Europe Conference & Exhibition, DATE, 2022, pp. 802–807, http://dx.doi.org/10.23919/DATE54114.2022.9774557.
[5] S. Mukherjee, Architecture Design for Soft Errors, Morgan Kaufmann, Burlington, 2008, http://dx.doi.org/10.1016/B978-0-12-369529-1.50012-4.
[6] S. Di Mascio, A. Menicucci, E. Gill, G. Furano, C. Monteleone, Leveraging the openness and modularity of RISC-V in space, J. Aerosp. Inf. Syst. 16 (2019) 1–19, http://dx.doi.org/10.2514/1.I010735.
[7] ISO 16290:2013 Technology Readiness Levels (TRLs), URL https://www.iso.org/standard/56064.html.
[8] mars.nasa.gov, Rover Brains - NASA, URL https://mars.nasa.gov/mars2020/spacecraft/rover/brains/.
[9] Radiation-hardened electronics, URL https://www.baesystems.com/en/product/radiation-hardened-electronics.
[10] Satellite Database | Union of Concerned Scientists, URL https://www.ucsusa.org/resources/satellite-database.
[11] S. Gupta, N. Gala, G.S. Madhusudan, V. Kamakoti, SHAKTI-f: A fault tolerant microprocessor architecture, in: 2015 IEEE 24th Asian Test Symposium, ATS, 2015, pp. 163–168, http://dx.doi.org/10.1109/ATS.2015.35.
[12] D.A. Santos, L.M. Luza, C.A. Zeferino, L. Dilillo, D.R. Melo, A low-cost fault-tolerant RISC-V processor for space systems, in: 2020 15th Design & Technology of Integrated Systems in Nanoscale Era, DTIS, 2020, pp. 1–5, http://dx.doi.org/10.1109/DTIS48698.2020.9081185.
[13] M. Olivieri, F. Menichelli, A. Mastrandrea, A. Cheikh, F. Vigli, L. Blasi, C.L. Blasi, The RISC-V Klessydra Orbital Lab project, URL https://indico.esa.int/event/323/contributions/5048/attachments/3749/5205/16.20_-_The_RISC-V_Klessydra_Orbital_Lab_project.pdf.
[14] I. Marques, C. Rodrigues, A. Tavares, S. Pinto, T. Gomes, Lock-V: A heterogeneous fault tolerance architecture based on arm and RISC-V, Microelectron. Reliab. 120 (2021) 114120, http://dx.doi.org/10.1016/j.microrel.2021.114120.
[15] Soft Error Mitigation (SEM) Core, URL https://www.xilinx.com/products/intellectual-property/sem.html.
[16] lowRISC: Collaborative open silicon engineering, URL https://lowrisc.org/.
[17] A. Ramos, J.A. Maestro, P. Reviriego, Characterizing a RISC-V SRAM-based FPGA implementation against single event upsets using fault injection, Microelectron. Reliab. 78 (2017) 205–211, http://dx.doi.org/10.1016/j.microrel.2017.09.007.
[18] Z. Mohseni, P. Reviriego, Reliability characterization and activity analysis of lowRISC internal modules against single event upsets using fault injection and RTL simulation, Microprocess. Microsyst. 71 (2019) 102871, http://dx.doi.org/10.1016/j.micpro.2019.102871.
[19] Rocket Chip Generator, URL https://github.com/chipsalliance/rocket-chip. original-date: 2014-09-12T07:04:30Z.
[20] L.A. Aranda, N.-J. Wessman, L. Santos, A. Sánchez-Macián, J. Andersson, R. Weigand, J.A. Maestro, Analysis of the critical bits of a RISC-V processor implemented in an SRAM-based FPGA for space applications, Electronics 9 (1) (2020) 175, http://dx.doi.org/10.3390/electronics9010175, Number: 1 Publisher: Multidisciplinary Digital Publishing Institute.

[21] J. Laurent, V. Beroulle, C. Deleuze, F. Pebay-Peyroula, Fault injection on hidden registers in a RISC-V rocket processor and software countermeasures, in: 2019 Design, Automation & Test in Europe Conference & Exhibition, DATE, 2019, pp. 252–255, http://dx.doi.org/10.23919/DATE.2019.8715158.

[22] H. Cho, Impact of microarchitectural differences of RISC-V processor cores on soft error effects, IEEE Access PP (2018) 1, http://dx.doi.org/10.1109/ACCESS.2018.2858773.

[23] M.A. Aguirre, J.N. Tombs, A. Torralba, L.G. Franquelo, UNSHADES-1: An advanced tool for in-system run-time hardware debugging, in: P. Y. K. Cheung, G.A. Constantinides (Eds.), Field Programmable Logic and Application, in: Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 2003, pp. 1170–1173, http://dx.doi.org/10.1007/978-3-540-45234-8_146.

[24] J. Mogollón, H. Guzmán-Miranda, J. Nápoles, J. Barrientos, M. Aguirre, FTUNSHADES2: A novel platform for early evaluation of robustness against SEE, in: 2011 12th European Conference on Radiation and Its Effects on Components and Systems, 2011, pp. 169–174, http://dx.doi.org/10.1109/RADECS.2011.6131392.

[25] M.A. Aguirre, H. Guzmán-Miranda, J.M. Mogollón, J. Nápoles, J. Barrientos, L. Sanz, FT-Unshades2: High speed, high capacity fault tolerance emulation system, 2013, URL https://amstel.estec.esa.int/tecedm/website/conferences/PresentationDays/FTU_UoSeville.pdf.

[26] M. Aguirre, H. Miranda, J. Rojas, L. Sanz, P. Corpas, FT-UNSHADES2, the user friendly framework as interface for designer support, 2016, URL https://www.researchgate.net/publication/298397936_FT-UNSHADES2_the_User_Friendly_Framework_as_interface_for_designer_support.

[27] A.E. Wilson, M. Wirthlin, Neutron radiation testing of fault tolerant RISC-v soft processor on xilinx SRAM-based FPGAs, in: 2019 IEEE Space Computing Conference, SCC, 2019, pp. 25–32, http://dx.doi.org/10.1109/SpaceComp.2019.00008.

[28] A.E. Wilson, S. Larsen, C. Wilson, C. Thurlow, M. Wirthlin, Neutron radiation testing of a TMR VexRiscv soft processor on SRAM-based FPGAs, IEEE Trans. Nucl. Sci. 68 (5) (2021) 1054–1060, http://dx.doi.org/10.1109/TNS.2021.3068835, Conference Name: IEEE Transactions on Nuclear Science.

[29] A.E. Wilson, M. Wirthlin, Fault injection of TMR open source RISC-V processors using dynamic partial reconfiguration on SRAM-based FPGAs, in: 2021 IEEE Space Computing Conference, SCC, 2021, pp. 1–8, http://dx.doi.org/10.1109/SCC49971.2021.00008.

[30] J. Dong, L. Xi, M. Yu, Z. Zhang, Software simulation error injection in RAM on RISC-v of PolarFire FPGA, in: 2019 IEEE 19th International Conference on Software Quality, Reliability and Security Companion, QRS-C, 2019, pp. 499–502, http://dx.doi.org/10.1109/QRS-C.2019.00095.

[31] 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, F.L. Kastensmidt, Evaluating soft core RISC-V processor in SRAM-based FPGA under radiation effects, IEEE Trans. Nucl. Sci. 67 (7) (2020) 1503–1510, http://dx.doi.org/10.1109/TNS.2020.2995729, Conference Name: IEEE Transactions on Nuclear Science.

[32] J. Lowe-Power, A.M. Ahmad, et al., The gem5 simulator: Version 20.0+, 2020, http://dx.doi.org/10.48550/arXiv.2007.03152.

[33] QEMU, URL https://www.qemu.org/.

[34] J. Gava, V. Bandeira, F. Rosa, R. Garibotti, R. Reis, L. Ost, SOFIA: An automated framework for early soft error assessment, identification, and mitigation, J. Syst. Archit. 131 (2022) 102710, http://dx.doi.org/10.1016/j.sysarc.2022.102710.

[35] V. Bandeira, F. Rosa, R. Reis, L. Ost, Non-intrusive fault injection techniques for efficient soft error vulnerability analysis, in: 2019 IFIP/IEEE 27th International Conference on Very Large Scale Integration, VLSI-SoC, 2019, pp. 123–128, http://dx.doi.org/10.1109/VLSI-SoC.2019.8920378.

[36] N. Lodéa, W. Nunes, V. Zanini, M. Sartori, L. Ost, N. Calazans, R. Garibotti, C. Marcon, Early soft error reliability analysis on RISC-V, IEEE Lat. Am. Trans. 100 (2022) http://dx.doi.org/10.1109/TLA.2022.9878169.

[37] SiFive, SiFive HiFive Boards, URL https://www.sifive.com/boards.

[38] B. James, H. Quinn, M. Wirthlin, J. Goeders, Applying compiler-automated software fault tolerance to multiple processor platforms, IEEE Trans. Nucl. Sci. 67 (1) (2020) 321–327, http://dx.doi.org/10.1109/TNS.2019.2959975, Conference Name: IEEE Transactions on Nuclear Science.

[39] T. Marena, RISC-V: High performance embedded SweRV™ core microarchitecture, performance and CHIPS alliance, 2019, URL https://riscv.org/wp-content/uploads/2019/04/RISC-V_SweRV_Roadshow-.pdf.

[40] RISC-V, URL https://riscv.org/.

[41] VeeR EH1 RISC-V Core, URL https://github.com/chipsalliance/Cores-VeeR-EH1.

[42] Z.Z. Bandic, R. Golla, – Architecture and microarchitecture – performance, 2019, URL https://riscv.org/wp-content/uploads/2019/12/12.11-14.20a3-Bandic-WD_SweRV_Cores_Roadmap_v4SCR.pdf.

[43] S. Harris, D. Chaver, RVfpga: Understanding Computer Architecture, URL https://university.imgtec.com/rvfpga-download-page-en/.

[44] O. Kindgren, VeeRwolf, URL https://github.com/chipsalliance/VeeRwolf.

[45] D. Leon, FPGA implementation of an AD-HOC RISC-V system-on-chip for industrial IoT, URL https://eprints.ucm.es/id/eprint/62106/.

[46] B. Guaitoune Akdi, D. Ledesma Ventura, RISC-V Processor implementation over fpga and memory encryption, URL https://eprints.ucm.es/id/eprint/67611/.

[47] M. Barbirotta, A. Cheikh, A. Mastrandrea, F. Menichelli, M. Olivieri, Design and evaluation of buffered triple modular redundancy in interleaved-multi-threading processors, IEEE Access 10 (2022) 126074–126088, http://dx.doi.org/10.1109/ACCESS.2022.3225975, Conference Name: IEEE Access.

[48] J.C. Fabero, G. Korkian, F.J. Franco, G. Hubert, H. Mecha, M. Letiche, J.A. Clemente, SEE sensitivity of a COTS 28-nm SRAM-based FPGA under thermal neutrons and different incident angles, Microprocess. Microsyst. 96 (2023) 1–10, http://dx.doi.org/10.1016/j.micpro.2022.104743.

[49] S. Beamer, A case for accelerating software RTL simulation, IEEE Micro 40 (4) (2020) 112–119, http://dx.doi.org/10.1109/MM.2020.2997639, Conference Name: IEEE Micro.

[50] M. Emami, S. Kashani, K. Kamahori, M.S. Pourghannad, R. Raj, J.R. Larus, Manticore: Hardware-accelerated RTL simulation with static bulk-synchronous parallelism, 2023, http://dx.doi.org/10.48550/arXiv.2301.09413.

[51] D. Báscones, C. Gonzalez, D. Mozos, A real-time FPGA implementation of the CCSDS 123.0-B-2 standard, IEEE Trans. Geosci. Remote Sens. 60 (2022) 1–13, http://dx.doi.org/10.1109/TGRS.2022.3160646, Conference Name: IEEE Transactions on Geoscience and Remote Sensing.

[52] RISC-V Specifications, URL https://riscv.org/technical/specifications/.

[53] R. Leveugle, A. Calvez, P. Maistri, P. Vanhauwaert, Statistical fault injection: Quantified error and confidence, in: Automation & Test in Europe Conference & Exhibition 2009 Design, 2009, pp. 502–506, http://dx.doi.org/10.1109/DATE.2009.5090716.

[54] S. Mukherjee, C. Weaver, J. Emer, S. Reinhardt, T. Austin, A systematic methodology to compute the architectural vulnerability factors for a high-performance microprocessor, in: Proceedings. 36th Annual IEEE/ACM International Symposium on Microarchitecture, 2003. MICRO-36, 2003, pp. 29–40, http://dx.doi.org/10.1109/MICRO.2003.1253181.

[55] M. Maniatakos, C. Tirumurti, A. Jas, Y. Makris, AVF analysis acceleration via hierarchical fault pruning, in: 2011 Sixteenth IEEE European Test Symposium, 2011, pp. 87–92, http://dx.doi.org/10.1109/ETS.2011.42, ISSN: 1558-1780.

[56] B. Goossens, Installing and using the RISC-V tools, 2023, pp. 87–103, http://dx.doi.org/10.1007/978-3-031-18023-1_3.

[57] T.C.C. for Space Data Systems, Low-complexity lossless and near-lossless multispectral and hyperspectral image compression, 2021, URL https://public.ccsds.org/Pubs/123x0b2c3.pdf.

[58] Z. Lei, F. Cai, J. Zhou, Z. Guo, A floating-point unit architecture based on SweRV EH1 core, in: 2022 IEEE 16th International Conference on Anti-Counterfeiting, Security, and Identification, ASID, pp. 1–5, http://dx.doi.org/10.1109/ASID56930.2022.9995796.

[59] A. Perea Rodríguez, Extensiones de punto flotante para el core SweRV EH1, URL https://hdl.handle.net/20.500.14352/87895.

[60] S. Mach, F. Schuiki, F. Zaruba, L. Benini, FPnew: An open-source multiformat floating-point unit architecture for energy-proportional transprecision computing, IEEE Trans. Very Large Scale Integr. (VLSI) Syst. 29 (4) 774–787, http://dx.doi.org/10.1109/TVLSI.2020.3044752, Conference Name: IEEE Transactions on Very Large Scale Integration (VLSI) Systems.

[61] Specification Status - RISC-V International, URL https://wiki.riscv.org/display/HOME/Specification+Status.

[62] T. Fritzmann, G. Sigl, J. Sepúlveda, RISQ-v: Tightly coupled RISC-V accelerators for post-quantum cryptography, IACR Trans. Cryptogr. Hardw. Embed. Syst. (2020) 239–280, http://dx.doi.org/10.13154/tches.v2020.i4.239-280.

[63] P. Nannipieri, S. Di Matteo, L. Zulberti, F. Albicocchi, S. Saponara, L. Fanucci, A RISC-V post quantum cryptography instruction set extension for number theoretic transform to speed-up CRYSTALS algorithms, IEEE Access 9 (2021) 150798–150808, http://dx.doi.org/10.1109/ACCESS.2021.3126208, Conference Name: IEEE Access.

[64] A.C. Canto, J. Kaur, M.M. Kermani, R. Azarderakhsh, Algorithmic security is insufficient: A comprehensive survey on implementation attacks haunting post-quantum security, 2023, http://dx.doi.org/10.48550/arXiv.2305.13544.

[65] J. Kaur, A.C. Canto, M.M. Kermani, R. Azarderakhsh, A comprehensive survey on the implementations, attacks, and countermeasures of the current NIST lightweight cryptography standard, 2023, http://dx.doi.org/10.48550/arXiv.2304.06222.

[66] A. Sarker, M.M. Kermani, R. Azarderakhsh, Error detection architectures for ring polynomial multiplication and modular reduction of ring-LWE in \boldsymbol\frac\mathbbZ/p\mathbbZ[x]x^n+1 benchmarked on ASIC, IEEE Trans. Reliab. 70 (1) (2021) 362–370, http://dx.doi.org/10.1109/TR.2020.2991671, Conference Name: IEEE Transactions on Reliability.

[67] M. Mozaffari Kermani, A. Reyhani-Masoleh, A high-performance fault diagnosis approach for the AES subbytes utilizing mixed bases, 2011, http://dx.doi.org/10.1109/FDTC.2011.11, Journal Abbreviation: Proceedings - 2011 Workshop on Fault Diagnosis and Tolerance in Cryptography, FDTC 2011 Pages: 87 Publication Title: Proceedings - 2011 Workshop on Fault Diagnosis and Tolerance in Cryptography, FDTC 2011.

Daniel León received his Computer Engineering and MSc. on the Internet of Things degrees from the Complutense University of Madrid (UCM), Spain, where he is currently working on a Ph.D. thesis on the radiation effects on RISC-V processors for space missions. He also holds post-graduate degrees in Business Management from Industrial Organization School in Madrid (EOI) and Humanities from Universidad Francisco de Vitoria of Madrid (UFV). For the last six years, he has been part of the Francisco de Vitoria University, serving as professor and deputy director of the computer engineering degree within the Higher Polytechnic School. Between 1991 and 2004, he worked as an engineer, consultant, and director at Hewlett-Packard, Vodafone, EY, Philips and IBM before becoming a managing partner and major stakeholder of Optiva Media from 2004 to 2012. Since 2013 he has served as business advisor in several boards of directors. During his career, he has been based in Madrid/Spain, Heidelberg/Germany, Costa Mesa/CA/United States of America, and Sydney/Australia.

Juan C. Fabero received a B.S. in physics and a Ph.D. in Computer Science from the Complutense University of Madrid (UCM). He is currently an Associate Professor with the Computer Architecture and Automation Department, UCM, in the GHADIR Research Group on dynamically reconfigurable architectures. His research interests include design automation, computer architecture, reconfigurable computing, and computer networks. He has been the assistant director of the Computer Architecture and Automation Department, UCM, from 2018 until now.

Juan A. Clemente received his degree in computer science and his Ph.D. degree from Complutense University of Madrid (UCM), Madrid, Spain, in 2007 and 2011, respectively. He is an Associate Professor with the Computer Architecture Department, UCM, and a Researcher with the GHADIR Research Group. His research interests include studying single-event effects tolerance of digital circuits, especially commercial- off-the-shelf memories, and their use in harsh environments, such as space. For this research, he collaborates with the TIMA Laboratory, Grenoble-Alpes University, Grenoble, France, and with the ONERA (the French Aerospace Lab), Toulouse, France.