A New Reliability Analysis of RISC-V Soft Processor for Safety-Critical Systems

A New Reliability Analysis of RISC-V Soft Processor for Safety-Critical Systems

Giorgio Cora, Corrado De Sio, Daniele Rizzieri, Sarah Azimi, Luca Sterpone,
Department of Control and Computer Engineering
Politecnico di Torino
Torino, Italy

Abstract— **RISC-V soft processors are attractive for various applications, including mission-critical ones, thanks to their reduced costs and high flexibility. Despite their growing popularity, reliability analysis of such platforms is still in an early stage, mainly relying on system-level analysis only, leaving module-level assessment unexplored. Such limitations hinder the development of mitigation strategies that could effectively focus on vulnerabilities within a RISC-V soft processor system. We propose a methodology for evaluating the module-wise reliability of a RISC-V soft processor based on fine-grained fault injection, custom layout placement, and fault analysis. Through this approach, we can provide insights into the critical elements of the processor, identifying the most susceptible to faults, both at the module and system levels. The presented results enhance comprehension of weak points within the processor, paving the way for creating robust and dependable RISC-V systems.**

Keywords— Fault Injection, FPGA, Reliability, RISC-V, Robustness, Single Event Upset, Soft Errors.

I. INTRODUCTION

In the last few years, RISC-V-based architectures have gained widespread attention in many application domains, including but not limited to embedded systems, edge computing, aerospace, and avionics [1]. The high level of flexibility and customization they provide make them attractive even for safety-critical applications. In the safety-critical domains, ensuring the reliability and robustness of such systems is a primary concern. Indeed, in safety-critical applications such as aerospace, automotive, and medical, system failures can have catastrophic consequences. Therefore, a comprehensive understanding of the reliability characteristics of RISC-V processors is imperative to develop strategies that can effectively mitigate potential failures and improve the overall dependability of systems.

Field Programmable Gate Arrays (FPGAs) are the primary candidates for achieving high performance with low costs in many application fields, mainly when characterized by low volumes, like in the space industry. FPGA devices are characterized by reprogrammable resources, such as Look-Up Tables (LUTs), Programmable Interconnection Points (PIPs), high-speed I/O, and Block RAMs (BRAMs). Such platforms represent an appealing solution for achieving high performance through custom hardware acceleration. Additionally, their hardware re-programmability is vital for extending mission lifetime and preventing obsolescence. Moreover, by implementing soft microprocessors, they can be enhanced to support software programmability. A soft microprocessor is a synthesizable processor system that can be implemented in the programmable hardware of FPGA-like systems. In particular, RISC-V-based soft processors have gained increasing popularity in industry and academia thanks to the broad support in software toolchains and libraries resulting from RISC-V ISA free and permissive license. FPGA applications can exploit soft processors, particularly RISC-V-based ones, to combine software flexibility with a custom hardware solution. Additionally, soft processors can be customized to meet specific requirements, such as higher reliability or performance. All these features made FPGA and RISC-V soft processors an attractive solution in many safety-critical domains.

Nevertheless, such flexibility must be considered along with reliability concerns. Since the circuit implemented in the programmable hardware of SRAM-based FPGA is defined by the content of a configuration memory (CRAM), the corruption of the CRAM content can produce unexpected modifications in the hardware architecture of the soft processor implemented in the programmable hardware. For instance, hardware faults such as logic gate corruption or stuck-at interconnection can occur due to single-bit flips in the configuration memory. Such hardware architectural changes persist and accumulate in the soft processor structure until they are corrected by rewriting the configuration memory content with the correct value. Single-event upsets (SEUs) are the primary source of data corruption for SRAM-based FPGA systems operating at elevated altitudes and a primary source of failure in space applications [2][3].

A SEU is an undesired modification of the value stored in a memory element caused by the physical interaction with ionizing radiation. SEU occurs already at sea level, and many factors, such as altitude, solar activity, and operational environment, influence their rate. Configuration memory corruption due to SEU is a well-known issue affecting the reliability of programmable hardware systems and a primary concern for safety-critical applications. Many mitigation methodologies have been proposed to improve the robustness of soft processors against these events. However, N-modular redundancy, where the design can be replicated at a different granular level to detect and correct deviation from the nominal behavior of the system, is the most commonly adopted. Even if this technique improves overall reliability, they have a cost in terms of used resources, power consumption, and performance, introducing a considerable overhead [4].

A. Problem Statement and Main Contribution

Despite the interest in RISC-V-based soft processors for safety-critical applications, the reliability evaluation of these systems is still in an early stage. Currently, research work focuses on system-level analyses that are unaware of the source of faults that caused the error that propagated to system-level. Even if such studies estimate the system's robustness, they provide limited observability and understanding of the mechanism that led to errors. Such a feature is hindered because the vendor does not make available the relationship between configuration memory bits and hardware resources used to implement the electronic circuits. This lack of knowledge prevents the possibility of relating faults with elements and modules of hardware architecture precisely. Since it has been proven that relying only on the software level can deliver distorted assessment [5], this lack of observability prevents an accurate detection of

979-8-3503-5934-3/24/$31.00 ©2024 IEEE

weak points in the microprocessor design and implementation, forcing designers to rely on higher-level approaches to assess the criticality of modules, such as simulation of a register-transfer-level (RTL), which nevertheless operates with a greater level of abstraction.

The main contribution of this work is to propose a new methodology suitable for conducting sensitivity analysis of complex circuits implemented on FPGA, such as RISC-V systems. The main novelty is a new complete evaluation flow that combines features offered by vendors and academic tools to perform a comprehensive analysis to identify the more vulnerable parts of a design. The proposed approach has several advantages. Firstly, it allows identifying with high precision the source of a fault in terms of resources and modules, providing valuable information for adopting fine-grained mitigation strategies. Secondly, it operates and emulates faults directly on the actual hardware device, providing significant speed-up compared to software simulation approaches and results closer to reality thanks to the involvement of the actual system. Since it does not require highly specialized gear or facilities, the method is easy to integrate into a traditional development flow, allowing the adoption of such analysis during development, prototyping, and preliminary research in preparation for a more demanding approach such as radiation testing.

This work is organized as follows. Section II describes the state of the art. Section III presents the proposed methodology. Finally, a case study and conclusions are presented in sections IV and V, respectively.

II. State of the Art

Recently, RISC-V-based processor systems have undergone several investigations for their adoption in space environments. The authors of [1] and [6] provided preliminary considerations about the advantages and limitations of applying such technology in radiation environments, such as near-earth orbits and outer space. While the sensitivity of the FPGA devices to SEUs in CRAM has been extensively investigated [7][8][12], the evaluation of RISC-V systems implemented in programmable hardware is still in an early stage. Indeed, the increasing complexity of devices and design complexity makes it challenging to effectively analyze the reliability characteristics of complex systems, such as RISC-V soft processors. A few works proposed reliability analysis for RISC-V design implemented on SRAM-based FPGA devices. Such studies are mainly based on radiation testing and fault simulation or emulation. Many works proposed neutron testing of RISC-V architectures [9]-[11]. Even if radiation testing is the primary methodology for evaluating the fault tolerance of devices and systems for space applications, it has some limitations. In particular, it requires a specialized facility, limiting its adoption during development phases. Moreover, since fault locations and occurrences are not strictly controlled, it provides a realistic assessment of the robustness but a minor understandability. For instance, it is complex to comprehensively understand what underlying effects led to specific errors at the system level or evaluate only a particular system component. For this reason, fault emulation is also widely adopted as a supplementary analysis, and it has proven to provide a reasonable estimation of system robustness [11][13]. It is also more controllable since it can target FPGA independently by other system elements, such as on-board elements, which is usually harder to achieve in radiation experiments. For this reason, it is also adopted for comparing radiation-hardened approaches [14][15].

Despite these interesting results, all fault injection approaches in the literature have one substantial limitation. The complexity of last-generation devices and systems and the *security by obscurity* policy adopted by FPGA companies have hindered most attempts to evaluate the reliability of specific modules composing a design on FPGA. As a result, fault emulation is carried out as a black-box analysis without methodology or tools for obtaining real controllability and understandability of the fault process, resulting in the impossibility of relating the fault emulated in configuration memory with the component of the circuit involved. Consequently, we have a good system reliability assessment but cannot identify which module or part of the circuit is more sensitive. As a result, mitigation approaches must be usually applied without precise targeting, leading to inefficiencies in addressing vulnerabilities or significant overhead.

III. Reliability Analysis Methodology

The methodology proposes a reliability evaluation flow that exploits isolation flow, bitstream decoding tools, and fault injection methodology to target and analyze single modules of complex systems, such as soft processors, independently and in greater detail. Such analysis produces a complete overview of the system's most sensitive elements, which is impossible to achieve with analyses commonly adopted during the evaluation of the system implemented on FPGA, where faults are injected in configuration memory without fundamental knowledge of the elements targeted. Additionally, the approach permits an easy comparison between different module implementations, such as using different mitigation approaches. The methodology consists of three phases. At first, modules to be analyzed are identified, and a custom placement is performed to facilitate mapping between configuration memory bits and modules under test. Secondly, configuration memory data and layout are analyzed to associate fault locations and modules. Finally, the fault injection process is performed by emulating faults in the device's configuration memory.

A. Isolated Layout Generation

FPGA development flow is characterized by a place-and-route phase where the synthesized netlist is mapped to specific resources by the tools. The Isolation Design Flow (IDF) is a fault-tolerant methodology that has proven to improve the reliability of design implemented on FPGA systems [16]-[18]. The isolation design flow consists of controlling the placement of the modules of a system within the FPGA fabric to create isolation of different functional modules to limit fault propagation and ease fault detection. Such a result is obtained by placement constraints applied during the flow.

The methodology's first step consists of forcing the isolation of the modules in the FPGA fabric for reliability evaluation purposes. FPGA development tools – e.g., AMD-Xilinx Vivado – support placement containers called *placement blocks* (Pblocks) to ease the isolation placement. To better understand the module-wise isolation concept, in Fig. 1, an implementation comparison between an unconstrained soft processor and a module-wise constrained one is reported. In Fig. 1(a), the unconstrained policy places modules unisolated, sharing physical location tiles and sometimes resources. In Fig. 1(b), six modules have been isolated for evaluation, while the fault injector and other modules that do not need to be evaluated have been grouped separately (cyan-colored). During this phase, the modules to be evaluated must be isolated from the others. Isolation is

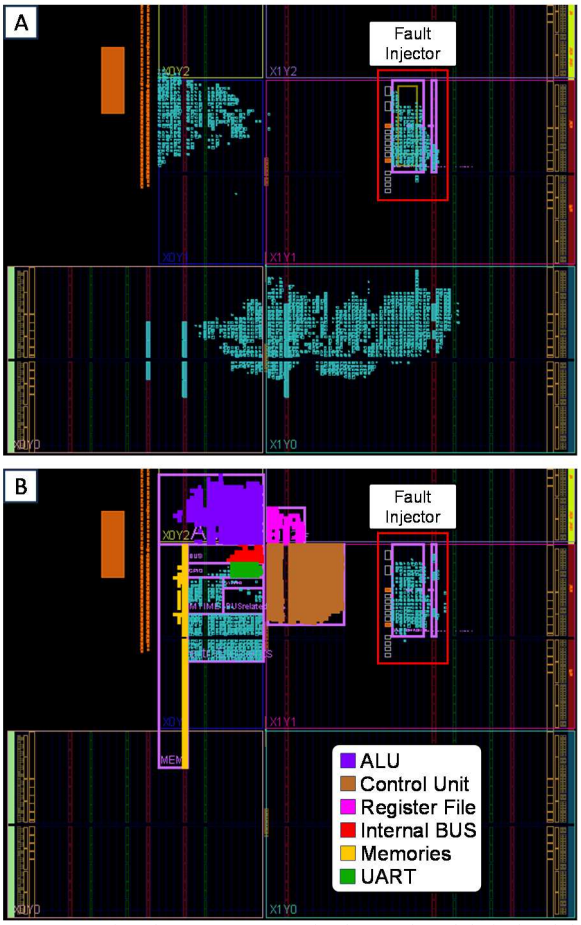

Fig. 1. Comparison between unconstrained (a) and module-isolated (b) soft-processor implementation.

needed for easing fault location identification carried out during the next step and limiting fault propagation to better assess each module's robustness.

B. *Fault Location Identification*

The current phase of the methodology is dedicated to identifying the configuration memory bits associated with each specific module. The overall flow and fault location identification phase, including the generation of isolated layout, is represented in Fig. 2 and elaborated in the current section. As a result of this phase, for each considered core module, a report is produced containing the coordinates – i.e., CRAM frame and bit offset – corresponding to all the Essential Bits of a target Pblock. The Essential Bits are defined as the bits that, if corrupted, may produce a functional error. Indeed, SRAM-based programmable devices use configuration memory to store information about the hardware architecture implemented in the device fabric. However, due to the typical FPGA architecture, the circuit uses only a subset of the resources composing the FPGA fabric when a design is implemented in the device. Consequently, only some configuration memory bits will be a source of errors when corrupted. For instance, in [19], it is reported that 5% of bits, on average, cause a functional soft error, and in the worst case, never more than 10% of the upsets cause a functional error. Such bits are referred to as critical bits and are a subset of the essential bits, and the only way to know if an essential bit is a critical bit is experimental. Modern devices are characterized by large configuration memory, which makes it unfeasible to evaluate each bit exhaustively. Such huge fault injection space (more than 10^8 for the smallest devices) challenges the comprehensive reliability evaluation of safety-critical systems.

The only approach currently provided by vendors to cope with such a problem is the combination of Essential Bits Data (EBD) and Essential Bits Content (EBC) files. EBD file is an instrumentation file that contains information for the AMD Soft-Error Mitigation (SEM) IP Core [20]. AMD SEM-IP Core is a mitigation core usually implemented within the FPGA fabric that performs configuration memory scrubbing to avoid fault accumulation. By loading an EBD file in the configuration step of SEM-IP Core, the SEM-IP Core is aware of which bits are essential and must be corrected if corrupted. EBC file contains the golden value for such memory configuration location. We must emphasize that not all the essential bits will cause an error if corrupted. As reported by the vendors and experimentally validated, only some essential bits, named critical bits, will introduce functional errors in the system. However, whether an essential bit is critical can be Aare provided in a proprietary format only to be used by SEM-IP and does not include any information about bits associated with a specific location or modules.

Very recently, a few tools have been presented aiming to solve the problem of mapping configuration memory data and resources. Even if they cannot create a functioning bitstream from scratch, they can map specific bits of the configuration memory to specific resources of the FPGA fabric [21]-[23]. With the support of these tools, it has been possible to identify a section of the configuration memory associated with each Pblock. The PyXEL toolkit [22] has been used in our flow to identify the configuration memory section to match the Essential Bits Content (.EBC) and consequently associate essential bits location to each module. To clarify further, since we can relate essential bits with the configured resource in the FPGA fabric using academic tools, we can also identify the PBlock enclosing such resource and, consequently, the module. As a result of isolation placement and fault location identification phases, we have a precise mapping of CRAM bits that are essential and labeled with a specific module of the system hardware architecture. Moreover, since this information has been obtained directly from analysis of the configuration memory file, essential bits data file, and essential bits content file, we have also coordinates of such bits in different coordinates that can support different fault emulation methodologies, such as through JTAG (i.e., using a configuration file) or through ICAP (i.e., instrumenting SEM-IP core for modifying specific bits in configuration memory).

C. *Fault Emulation*

During the last phase, the fault injection experiments are executed. Based on the specific device architecture, the fault injection analysis can be conducted with different

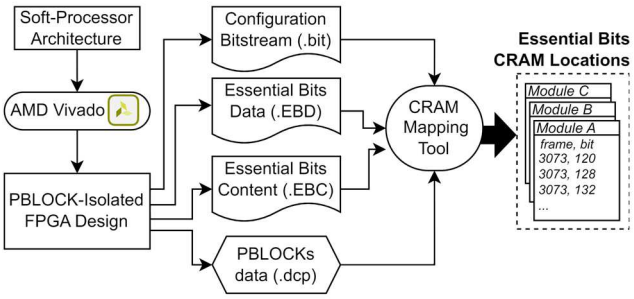

Fig. 2. Flowchart showing the phases to generate the fault locations

methodologies exploiting the frame-and-bit fault locations

obtained as the output of the previous phase. However, since the proposed method incorporates the concept of essential bits, the use of SEM IP Core is preferred, especially since such a fault emulation procedure is way faster than reconfiguring the device with a faulty bitstream, resulting in an injection time of a few hundred milliseconds compared to seconds. The flow of the fault emulation experiments is represented in Fig. 3.

The SEM IP Core is commonly used to perform continuous configuration memory scrubbing. Still, it can be used to alter the content of the CRAM itself, practically emulating Single Event Upset in the configuration memory. The SEM-IP core uses a proprietary addressing format to refer to bits in the configuration memory content. Starting from the CRAM fault locations obtained in the previous phases, it is possible to use CRAM mapping tools to create a fault location list associated with a module to instrument SEM IP Core. A Python-based experiment manager on the host computer can orchestrate the system to perform the reliability experiment. Firstly, the experiment manager uses the SEM-IP to corrupt the value of a configuration memory bit of the fault location list to emulate the faults affecting the target module. Then, the benchmark applications are executed in the processor. The results are compared with the nominal ones, and a reliability report is updated accordingly. Before performing a new fault emulation, the configuration memory is restored to the original content by instrumenting the SEM-IP Core. Finally, the circuit is reset to return to the starting state, and a new fault emulation experiment is started. Additionally, this approach is fully scalable on more devices since the fault location list can be trivially assigned to different devices, and the experiment

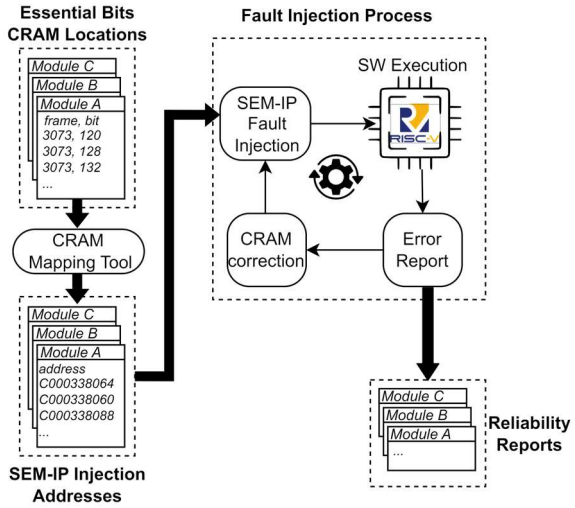

Fig. 3. Flowchart showing the fault injection procedure.

manager can easily control multiple devices.

IV. Case of Study: NEORV32

The flexibility of the proposed methodology allows it to be applied to different kinds of complex systems. In particular, any architecture that can be divided into separate functional units is suitable for such an analysis approach. A RISC-V system is the proposed case of study.

TABLE I. FPGA Resources Utilization of Target modules

Soft-Processor Module	FPGA Resources Utilization			
	LUT	FF	BRAM	DSP
Arithmetic Logic Unit (ALU)	920	797	0	2
Control Unit (CU)	1346	644	0	0
Register File (RF)	175	0	1	0
Internal Bus (BUS)	49	103	0	0
Internal Memories (MEMs)	38	4	12	0
Serial I/O Module (UART)	74	146	0	0

TABLE II. Essential Bits of Target Modules

Soft-Processor Module	Essential Bits	% of Total Essential Bits	% Total CRAM Bits
Arithmetic Logic Unit (ALU)	183,527	20.26	0.57
Control Unit (CU)	237,532	26.22	0.73
Register File (RF)	28,414	3.13	0.09
Internal Bus (BUS)	17,500	1.93	0.05
Internal Memories (MEMs)	43,207	4.76	0.13
Serial I/O Module (UART)	19,506	2.15	0.06
Total	529,676	58.45	1.64

A. Design Under Test

As the design under test, the NEORV32 Processor has been selected [24]. NEORV32 is a customizable microcontroller-like system on chip (SoC). Its architecture supports minimal core features composing the Central Processing Unit. It can be optionally integrated with several optional modules, permitting a broad set of configurations and many application scenarios. The NEORV32 is fully defined in behavioral and platform-independent VHDL and Verilog.

For this reason, it can be implemented into nearly any FPGA system. It supports both bare-metal and Real-Time Operating System (RTOS) executions. The CPU has been implemented using the *C*, *M*, *Zfinx*, *Zicntr*, and *Zihpm* instruction set extensions, thus allowing the following computing features: compressed instructions, multiplication and division, floating-point operations, and Control and Status Registers (CSRs) management functions. It is worth emphasizing that multiplication and division operations would be generally executed as a sequence of add/sub and shift computations when the M extension is not included. Apart from the CPU minimal set of modules, a small set of optional ones have been chosen and integrated to permit fast and straightforward execution of the selected benchmark applications. DMEM and IMEM modules implement the processor's internal data and instruction memories. Since the application benchmarks are run on a bare-metal system, memories are pre-initialized during the synthesis phase to reduce the experiment. The instruction memory also contains instructions to automatically boot the system once the reset is deactivated. Finally, a UART module provides a serial interface for output communication.

Six modules of the NEORV32 have been selected for the experimental analysis, and the isolated layout is reported in Fig. 1. They are the following:

- The Arithmetic Logic Unit (ALU) executes the arithmetic operations on the data from the register file and addresses computation for jumps, function calls, and branches. In our case study, we included the extensions to execute floating-point operations, multiplications, and division other than base operations such as addition, subtractions, and shift.
- The Control Unit (CU) controls the instruction fetching and signals to pilot the system. It is divided into front-end and back-end parts. The latter is responsible for control and status registers management, while the front-end controls instruction fetching, branch management, and code execution.
- Register File (RF) comprises 32-entries synchronous memory element. The registers are mapped as the standard RISC-V specifications, but in this architecture, the program counter register is part of the front-end of the control unit.
- The Internal Bus (BUS) module manages load/store operations, data alignment, and memory interface protocols.
- Processor Memories (MEMs) module includes the processor's internal data and instruction memories.

- The UART Controller (UART) controls the UART peripheral used by the processor to communicate with external devices through UART serial communication.

The selected modules have been chosen to cover the whole functionality of the soft processor. The design has been synthesized and implemented on a Xilinx Artix-7 FPGA device embedded in a Zynq-7020 SoC. Information about the logic used by the device is reported in Table I.

B. Software Benchmark

One of the most critical elements to consider when evaluating a reliability experiment is how stimulating the system. Especially in the case of a processor system, different programs can stimulate different parts of the circuit, leading to different outcomes. Hence, a comprehensive set of benchmarks based on the final application of the system is usually adopted. For instance, a reliability analysis could use a memory-bound or computational-bound program based on the expected workload of the system. For our study, we adopted a suite of widely used benchmark programs: Whetstone, LINPACK, MatMul, and Dhrystone.

1. *Whetstone* is a performance benchmark performing arithmetic, floating-point, and logic operations such as matrix convolution and angle and root computations.
2. *LINPACK* benchmark is a variation of LINPACK-100. It solves systems of linear equations with large matrices of floating-point numbers.
3. *MatMul* is a simple benchmark performing matrix multiplication between large matrices of integers.
4. *Dhrystone* is a performance benchmark based on integer arithmetic, string operations, and memory accesses.

C. Experimental Analysis

The configuration data of Zynq-7020, a relatively small device, comprises about 32M bits. Table II reports information about configuration memory and essential bits associated with each module, resulting from the Fault Location Identification phase of the methodology. It can be noticed in Table II that the analyzed modules sum up about 60% of the essential bits of the design. Indeed, as can be seen from Fig. 1, some parts of the design that do not belong to the soft processor, such as the fault injection system and interface for the host computer, are implemented on the device. Such additional elements contribute to the number of essential bits, even if they do not belong to the design under test. However, thanks to the procedure exposed in the Fault Location Identification phase of the methodology, it has been possible to identify the essential bits only associated with the design under test. Such features assure a precise reliability analysis and a reduced fault injection time for exhaustive analysis. Considering around 1.6% of the total CRAM bits, we achieve 100% covering of the critical configuration bits.

Moreover, relying only on the essential bits feature (i.e., without module mapping) would have reduced the analysis accuracy and increased evaluation time, including 40% of fault locations unrelated to the design under test. We want to emphasize that such analysis is not permitted by other approaches that do not associate essential bits with the correspondent module. Finally, please notice that covering all configuration memory is unfeasible with any methodology, while covering the totality of essential bits using other methodologies will lead to a long fault injection campaign. For instance, adopting a common methodology for downloading corrupted configuration memory data through JTAG (about 5 seconds per download) will require about 50 days without considering software execution time. The proposed methodology, relying on bitstream mapping, essential bits, and SEM IP Core, can cover 100% of the critical locations in about one day if we do not consider software execution time.

TABLE III. EXHAUSTIVE ERROR RATE OF MODULES

Software	ALU(%)	CU(%)	RF(%)	BUS(%)	MEM(%)	UART(%)
Whetstone	14.92	12.57	9.76	13.04	5.11	12.00
LINPACK	12.89	10.93	9.84	12.85	5.00	11.86
MatMul	3.64	8.05	6.83	18.06	5.59	11.74
Dhrystone	2.58	7.64	6.12	17.65	4.85	11.68
Average	8.51	9.80	8.14	15.40	5.14	11.82

D. Experimental Results and Discussion

Using the methodology illustrated in the current work, we exhaustively analyzed the modules of the NEORV32. The analysis required about one day using six boards in parallel. Since all essential bits of each module have been evaluated exhaustively, the number of faults analyzed for each module is the number of essential bits reported in Table II.

In Table III, the overall error rate for each module and each software application is reported. By looking at the overall error rate values, it is clear that the module most prone to errors is the internal bus module (BUS). Notice that this is true if we consider each module as an independent entity; this information helps identify a weak module in the system. Additionally, the possibility of obtaining such insight into the reliability characteristic of each module allows for easing evaluation of different architectures that could use a different version for a module or mitigation approaches, information that is not available otherwise analyzing the system as a black box, i.e., without module mapping. Table IV reports a detailed classification of errors resulting from the experimental analysis, grouped into the following categories:

- *Halt*: The processor system is halted due to injected faults. A timeout on the host computer detects such category errors.
- *Silent Data Corruption (SDC)*: The software terminates correctly, but the computation presents errors.
- *I/O Errors*: The faults cause a malfunction, leading to badly formatted data on the system's output.

Table IV reports which error is associated with a fault in specific modules. For instance, the halt category is the most common among all the modules except ALU, where SDC is more common for some software. Observing distribution and considering the error rate of each module on the whole system is possible to focus on mitigation strategies for reaching

TABLE IV. ERROR DISTRIBUTION

	Error Type	Benchmark Applications			
		Whetstone (%)	LINPACK (%)	MatMul (%)	Dhrystone (%)
ALU	HALT	10.30	11.81	**53.71**	**60.27**
	SDC	**82.27**	**86.22**	33.97	37.24
	I/O Err.	7.44	1.97	12.32	2.49
CU	HALT	**74.90**	**76.53**	**86.20**	**89.95**
	SDC	22.38	22.54	10.46	9.28
	I/O Err.	2.71	0.92	3.34	0.77
RF	HALT	**51.66**	**56.15**	**73.07**	**84.19**
	SDC	39.62	40.52	16.12	9.55
	I/O Err.	8.72	3.33	10.81	6.27
BUS	HALT	**88.21**	**90.04**	**90.00**	**89.77**
	SDC	10.21	9.21	6.45	7.12
	I/O Err.	1.58	0.76	3.54	3.11
MEMs	HALT	**91.89**	**96.21**	**85.60**	**93.99**
	SDC	4.89	1.80	7.61	4.34
	I/O Err.	3.22	1.99	6.79	1.67
UART	HALT	**92.56**	**93.95**	**94.37**	**95.04**
	SDC	5.47	4.80	3.58	3.99
	I/O Err.	1.97	1.25	2.05	0.97

specific reliability goals. For instance, if preventing erroneous results is the primary concern, mitigation approaches should focus on the ALU. Differently, if availability is the most critical metric, the CU is probably the best candidate for mitigation efforts.

Finally, even if the analysis focuses on single modules, the system level can also be considered. In particular, it is worth noticing how, despite what would result from the module-wise analysis, nearly 48% of the faults that could affect the system come from the Control Unit, with 26% of the essential bits and an error rate of around 10% is the module with the most significant weight on the system reliability. An overall overview of the contribution of each module to faults is illustrated in Fig. 4. Please notice that the error rate is weighted on the number of essential bits. To clarify further, it represents the percentage of critical bits of the system belonging to a specific module, while Table III reports the error rate of the specific module without considering its size.

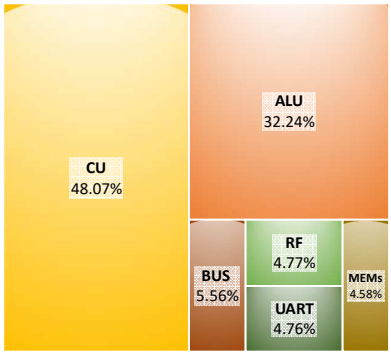

Fig.4. Sensitive bits cross section based on soft-processor modules.

V. Conclusions and Future Works

This work introduces a methodology exploiting fine-grained fault injection, customized layout placement, and comprehensive fault analysis. This approach allows us to evaluate the reliability of a RISC-V soft processor at the module level, pinpointing the modules most susceptible to faults, both within individual modules and across the entire system. The results of our study offer a deeper understanding of the weaknesses inherent in RISC-V soft processor systems. The proposed methodology provides a comprehensive analysis of the robustness characteristics of a complex system that can be exploited to improve its reliability efficiently.

References

[1] G. Furano, et al., "A European Roadmap to Leverage RISC-V in Space Applications," 2022 IEEE Aerospace Conference (AERO), Big Sky, MT, USA, 2022, pp. 1-7, doi: 10.1109/AERO53065.2022.9843361.

[2] H. Quinn, "Radiation effects in reconfigurable FPGAs", Semicond. Sci. Technol., vol. 32, no. 4, Apr. 2017

[3] E. Vacca, S. Azimi, L. Sterpone, “Failure rate analysis of radiation tolerant design techniques on SRAM-based FPGAs”, Microelectronics Reliability, Volume 138, 2022, 114778, ISSN 0026-2714, DOI: 10.1016/j.microrel.2022.114778.

[4] F. L. Kastensmidt, et al., "On the optimal design of triple modular redundancy logic for SRAM-based FPGAs," Design, Automation and Test in Europe, Munich, Germany, 2005, pp. 1290-1295 Vol. 2, doi: 10.1109/DATE.2005.229.

[5] G. Papadimitriou and D. Gizopoulos, "Demystifying the System Vulnerability Stack: Transient Fault Effects Across the Layers," 2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA), Valencia, Spain, 2021, pp. 902-915, doi: 10.1109/ISCA52012.2021.00075.

[6] S. Di Mascio et al., “The Case for RISC-V in Space,” in Applications in Electronics Pervading Industry, Environment and Society, S. Saponara and A. De Gloria, Eds., in Lecture Notes in Electrical Engineering. Cham: Springer International Publishing, 2019, pp. 319–325. doi: 10.1007/978-3-030-11973-7_37.

[7] T. Li et al., “Investigation into SEU Effects and Hardening Strategies in SRAM Based FPGA,” in 2017 17th European Conference on Radiation and Its Effects on Components and Systems (RADECS), Oct. 2017, pp. 1–5. doi: 10.1109/RADECS.2017.8696177.

[8] H. Michel, et al., "SEU fault classification by fault injection for an FPGA in the space instrument SOPHI," 2017 NASA/ESA Conference on Adaptive Hardware and Systems (AHS), Pasadena, CA, USA, 2017, pp. 9-15, doi: 10.1109/AHS.2017.8046353.

[9] B. E. Forlin, et al., "An unprotected RISC-V Soft-core processor on an SRAM FPGA: Is it as bad as it sounds?," 2023 IEEE European Test Symposium (ETS), Venezia, Italy, 2023, pp. 1-6, doi: 10.1109/ETS56758.2023.10174076.

[10] D. Santos, et al., "Neutron Irradiation Testing and Analysis of a Fault-Tolerant RISC-V System-on-Chip," in 2022 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), Austin, TX, USA, 2022 pp. 1-6. doi: 10.1109/DFT56152.2022.9962335

[11] Á. B. de Oliveira et al., "Evaluating Soft Core RISC-V Processor in SRAM-Based FPGA Under Radiation Effects," in IEEE Transactions on Nuclear Science, vol. 67, no. 7, pp. 1503-1510, July 2020, doi: 10.1109/TNS.2020.2995729.

[12] C. De Sio, S. Azimi, L. Sterpone and B. Du, "Analyzing Radiation-Induced Transient Errors on SRAM-Based FPGAs by Propagation of Broadening Effect," in *IEEE Access*, vol. 7, pp. 140182-140189, 2019, doi: 10.1109/ACCESS.2019.2915136.

[13] C. De Sio, S. Azimi and L. Sterpone, "FireNN: Neural Networks Reliability Evaluation on Hybrid Platforms," in *IEEE Transactions on Emerging Topics in Computing*, vol. 10, no. 2, pp. 549-563, 1 April-June 2022, doi: 10.1109/TETC.2022.3152668.

[14] Wali et al., "Analyzing the impact of the Operating System on the Reliability of a RISC-V FPGA Implementation," 2020 27th IEEE International Conference on Electronics, Circuits and Systems (ICECS), Glasgow, UK, 2020, pp. 1-4, doi: 10.1109/ICECS49266.2020.9294858.

[15] J. Andersson, “Development of a NOEL-V RISC-V SoC Targeting Space Applications,” in 2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W), Jun. 2020, pp. 66–67. doi: 10.1109/DSN-W50199.2020.00020.

[16] AMD, Isolation Design Flow for Xilinx 7 Series FPGAs or Zynq-7000 SoCs, (XAPP1335), 2023.

[17] E. Vacca, C. De Sio, and S. Azimi, “Layout-oriented radiation effects mitigation in RISC-V soft processor,” in Proc. 19th ACM Int. Conf. Comput. Frontiers, May 2022, pp. 215–220

[18] A. Portaluri, C. De Sio, S. Azimi and L. Sterpone, "A New Domains-based Isolation Design Flow for Reconfigurable SoCs," 2021 IEEE 27th International Symposium on On-Line Testing and Robust System Design (IOLTS), Torino, Italy, 2021, pp. 1-7, doi: 10.1109/IOLTS52814.2021.9486687.

[19] AMD, Device Reliability Report (UG116), 2023.

[20] AMD, Soft Error Mitigation Controller v4.1, (PG036), 2018

[21] I. Tuzov, et al., "BAFFI: a bit-accurate fault injector for improved dependability assessment of FPGA prototypes," 2023 Design, Automation & Test in Europe Conference & Exhibition (DATE), Antwerp, Belgium, 2023, pp. 1-6, doi: 10.23919/DATE56975.2023.10137300.

[22] C. De Sio, et al., “PyXEL: Exploring Bitstream Analysis to Assess and Enhance the Robustness of Designs on FPGAs,” in 2023 19th International Conference on Synthesis, Modeling, Analysis and Simulation Methods and Applications to Circuit Design (SMACD), Jul. 2023, pp. 1–4. doi: 10.1109/SMACD58065.2023.10192116.

[23] S. Kashani, M. Emami and J. R. Larus, "Bitfiltrator: A general approach for reverse-engineering Xilinx bitstream formats," 2022 32nd International Conference on Field-Programmable Logic and Applications (FPL), Belfast, United Kingdom, 2022, pp. 01-08, doi: 10.1109/FPL57034.2022.00039.

[24] S. Nolting and Contributors, “The NEORV32 RISC-V Processor.” Zenodo, Aug. 18, 2023. doi: 10.5281/zenodo.8260609