IEEE Access
Multidisciplinary ⁝ Rapid Review ⁝ Open Access Journal

Received 12 October 2022, accepted 28 November 2022, date of publication 1 December 2022, date of current version 7 December 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3225975

APPLIED RESEARCH

Design and Evaluation of Buffered Triple Modular Redundancy in Interleaved-Multi-Threading Processors

MARCELLO BARBIROTTA, ABDALLAH CHEIKH, ANTONIO MASTRANDREA, FRANCESCO MENICHELLI, AND MAURO OLIVIERI, (Senior Member, IEEE)
Department of Information Engineering, Electronics and Telecommunications (DIET), Sapienza University of Rome, 00184 Rome, Italy
Corresponding author: Marcello Barbirotta (marcello.barbirotta@uniroma1.it)

ABSTRACT Fault management in digital chips is a crucial aspect of functional safety. Significant work has been done on gate and microarchitecture level triple modular redundancy, and on functional redundancy in multi-core and simultaneous-multi-threading processors, whereas little has been done to quantify the fault tolerance potential of interleaved-multi-threading. In this study, we apply the temporal-spatial triple modular redundancy concept to interleaved-multi-threading processors through a design solution that we call Buffered triple modular redundancy, using the soft-core Klessydra-T03 as the basis for our experiments. We then illustrate the quantitative findings of a large fault-injection simulation campaign on the fault-tolerant core and discuss the vulnerability comparison with previous representative fault-tolerant designs. The results show that the obtained resilience is comparable to a full triple modular redundancy at the cost of execution cycle count overhead instead of hardware overhead, yet with higher achievable clock frequency.

INDEX TERMS Circuit faults, digital integrated circuits, fault detection, fault tolerant computing, field programmable gate arrays, microprocessors, multithreading, radiation hardening (electronics), redundancy, robustness.

I. INTRODUCTION

The probability of faults due to voltage glitches induced by ionizing particles in digital chips increase with the reduction of the minimum feature size and voltage margins, along with augmented statistical process variations [1], [2], [3]. The ability to manage circuit faults to preserve the system's functional safety, commonly denoted as fault tolerance (FT), is traditionally linked to space, avionics, and military worlds. Yet, it has been recognized as playing a central role in ground-level intelligent transportation systems [4], which increasingly rely on digital integrated circuits.

Radiation-hard fabrication technologies can be applied to a few high-budget products since they are orders of magnitude more expensive than commercial off-the-shelf (COTS) hardware components [5], [6]. Thus, fault-tolerant architecture design techniques (also known as radiation hardening by design) are key features in many embedded application domains that must employ COTS for commercial sustainability. In this context, the effective implementation of fault-tolerant soft-processor solutions in static RAM (SRAM) based FPGAs is a significant research target [7], [8].

Several fault-tolerant processor architecture approaches have been proposed in the literature to improve fault coverage (FC), that is, the percentage of faults that can be detected and resolved in the microarchitecture. These techniques are typically based on spatial redundancy (resource replication) and/or temporal redundancy (operation repetition), which always results in hardware overhead and/or speed degradation [9], [10].

Broadly speaking, the existing approaches can be divided into triple modular redundancy (TMR), which allows the detection and correction of faults using a one-over-three voting mechanism, and double modular redundancy (DMR), which allows fault detection and relies on a checkpoint

The associate editor coordinating the review of this manuscript and approving it for publication was Gerard-Andre Capolino.

restoration mechanism to recover the correct operation. Error correcting codes (ECC) are another form of redundancy typically reserved to protect memories and large register files.

In recent years, much work has been done to reliably exploit simultaneous multi-threading (SMT) and multi-core (MC) processors for implementing TMR and DMR, constituting the basis for many industrial fault-tolerant systems [11], [12]. The concept behind these works is replicating the same thread instead of executing different threads, obtaining an FT gain instead of a performance gain. In this study, we explore the use of interleaved multi-threading (IMT) to obtain fault-tolerant operations. IMT cores are single-issue in-order processors that fetch instructions from different execution threads in each clock cycle [13], [14]. They are interesting in low-cost embedded systems because they achieve relatively high performance by hiding data-dependency stalls providing a fence between read and write access to the register file [15]. As they support the bare-metal execution of multiple threads, they can potentially support FT through thread replication. However, they are less immediately usable for implementing FT support than SMT and MC architectures because they share some hardware resources among the threads, and the results of the replicated instructions are not simultaneously available.

Starting from an open-source Reduced Instruction Set Computer-V (RISC-V) IMT softcore (Klessydra-T03 [16]), we implemented a form of temporal-spatial redundancy using a set of precise hardware modifications having general validity, which we refer to as Buffered TMR. We quantified the FT performance of the so-obtained processor core, named Klessydra-fT03, by an extensive fault-injection campaign with single-event upset (SEU) faults, targeting all register bits in the architecture.

The proposed study is - to the best of our knowledge - the first detailed evaluation of the IMT execution scheme for implementing fault-tolerant processors, covering both design aspects and quantitative performance analysis. This work extends the concept presented in [17] through a deeper design discussion, a wide fault-injection simulation campaign with different benchmark application kernels, and a detailed comparison with other FT architectures taken from the literature, addressing power, frequency, hardware resource utilization, and FC.

The contributions of the proposed work with respect to the existing state of the art are the following:

- The work fills a gap in the literature, by analyzing the fault tolerance potential of IMT architectures, which is not covered by previous works in the field;
- It proposes and details the implementation of a technique that merges spatial and temporal redundancy, referred to as Buffered TMR;
- It quantifies the effectiveness of the proposed technique by means of an extensive fault-injection simulation campaign, in terms of Architecture Vulnerability Factor and Mean Work Between Failure;
- It discusses the comparison of the approach with other fault tolerance designs based on the same Instruction Set Architecture.

The rest of the article is organized as follows: Section II discusses how multi-threading has been exploited for FT in SMT and MC architectures and provides an overview of existing RISC-V FT cores that constitute the dataset for performance comparison with the proposed work; Section III discusses the proposed design solution; Section IV describes the verification methodology and environment, followed by the results in Section V and the comparison with existing works in Section VI; Section VII discusses the main outcomes, and Section VIII summarizes the conclusions.

II. RELATED WORKS

A. SIMULTANEOUS MULTI-THREADING PROCESSORS

One of the earliest approaches for exploiting redundant threads for FT is active redundant simultaneous multi-threading (AR-SMT), in which an active-stream thread saves its results in a buffer and a redundant-stream thread compares its results with the saved ones, producing an exception in the case of a mismatch [12]. Similarly, the simultaneous redundant thread (SRT) approach defines a leading thread and a follower (or trailing) thread, adopting a series of microarchitecture techniques to avoid performance stalls in load/store instructions and manage access to resources shared by the leader and follower threads [18]. The SRT-with-recovery (SRTR) approach [11], [12] extends SRT with instruction re-execution in the case of error detection. It also introduces microarchitecture enhancements, such as a register value queue (RVQ) to compare the value produced by the trailing thread with that produced by the leader thread, thereby saving the register file bandwidth. SRTR also provides dependence-based checking elision (DBCE), allowing only the last instruction in a dependence chain to be saved on the RVQ. In [19], a ‘‘safe-shuffle’’ approach is proposed to ensure that the leading and trailing threads use different hardware resources in order to address permanent hardware fault detection.

Other techniques have been introduced using partially redundant multi-threading to improve performance by limiting the number of redundant instructions executed by the processor. The authors of [20] propose Slipstream, an extension of AR-SMT in which a complex microarchitecture mechanism, by analyzing redundant thread execution possibly removes unnecessary instructions from the active thread because they are likely not to affect the state of the thread. In the case of state mismatch, the state of the active thread is recovered from the redundant thread. A similar concept is implemented in slice-based locality exploitation [21], which executes instructions from the leading thread, such as SRT, with a slice matrix that maintains the instruction trace and a mechanism that tries to delete instructions from the trailing thread by predicting the output. Finally, opportunistic transient fault detection [22] proposes a methodology to limit the performance degradation due to redundant execution by enabling or disabling the redundant thread.

B. MULTI-CORE PROCESSORS

The underlying idea in MC fault-tolerant architectures [11], [12] is the migration of all the techniques used in SMT cores to multiple cores. A particular implementation of MC fault tolerance is represented by lock-stepped cores [12], [23], in which synchronized processor replicas execute the same instruction in the same clock cycle (or with a fixed-time separation of a few cycles) and the results of each instruction are compared in real-time.

The chip-level redundant threading (CRT) approach [24], with and without recovery, implements the SRT technique using two cores, comparing them by sharing the branch prediction queue and store/load value queue via a dedicated bus. The Reunion approach [25] proposes a CRT that preserves the existing memory interface, coherence protocols, and consistency models, and uses a '"fingerprint" to compress the architecture status to reduce the comparison overhead, while authors in [10] developed an offline scheduler synthesis framework for MC processors executing real-time applications that, even in the presence of transient faults, can drive the system to a safe execution state. In [26] and [27], fault-tolerant multi-core and many-core processors are described and compared, while [28] proposes an FPGA methodology that combines two different cores with different performances.

Overall, the outcome of the analysis of the literature related to FT in multi-threading processors (SMT and MC) evidences a gap due to the absence of studies addressing IMT processors, which is the focus of the proposed work.

C. SOFTWARE FT IN MULTI-THREADING PROCESSORS

In both SMT and MC approaches, software redundancy methodologies have been extensively documented [12], such as error detection by duplicated instructions operating on distinct registers and memory regions or specific voting instructions plugged by the compiler. In software-implemented fault tolerance with recovery (SWIFT-R) [29], instructions are triplicated, with a simple voting mechanism in case of a fault on single instructions, whereas in triple redundancy using multiplication protection (TRUMP) [29], [30], an AN-encoding copy is used instead of an operation redundancy. The original value is multiplied by the constant value A to obtain the encoded copy. The recovery mechanism is activated when the encoded copy does not match the original values multiplied by a factor A. The work in [31] analyzes the hardened-by-replication at the software level, observing that these techniques improve marginally the reliability against in-memory SEUs, but they degrade the reliability against hardware architectural faults. In [32], a software approach is adopted, in which many cores are used to launch partial replicas of the same program.

Overall, a common characteristic of software-based approaches to FT is the implementation of temporal redundancy by modifying the software code to be run on the processor. In the proposed approach, temporal redundancy is intrinsically obtained by the automatic replication of the threads being executed in the microarchitecture, without the need for interventions on the application software code.

D. RISC-V FAULT-TOLERANT PROCESSOR CORES

When evaluating different microarchitectures for FT, care should be taken to ensure that the results may not be influenced by the different instruction-set architectures (ISA) on which the application benchmarks are compiled. Our experiments target the RISC-V ISA; thus, RISC-V-compliant fault-tolerant cores are the reference architectures for performance comparison. RISC-V is an open and extendable ISA that has gained growing interest in academia and industry since its introduction in 2010 [33]. Numerous RISC-V cores have been implemented for embedded system applications [16], [34], [35], and RISC-V has recently received attention for space applications, opening the way for the development of fault-tolerant RISC-V microarchitectures [6], [17], [28], [36], [37]. Table 1 summarizes the most representative RISC-V-based FT works, categorized by the applied techniques, hardened microarchitecture components, and verification/test methodology.

In [38], the authors present CEVERO, a RISC-V System-on-Chip that implements FT on a PULP platform [34]. It comprises two Ibex cores running in a lock-step execution, which are compared to each other via an FT hardware module that checks whether an error manifests in any executed instruction. The system is tested using a fault-injection (FI) hardware block capable of inserting bit flips in the hardware. The authors claim that the system can detect errors, but no numerical data about FT performance and fault coverage are reported. In [39], the SHAKTI-F architecture is reported, featuring spatial-temporal TMR for the ALU and ECC for registers and memories. This paper presents an interesting fault analysis and re-computation methodology that can classify faults as permanent or transient. Two levels of verification are reported, module level and system level, with an FI block that affects the single module or the entire system, respectively, with single-bit faults simulated by injecting bit flips in the registers between the pipeline stages. No exhaustive numbers are reported regarding fault coverage, but the authors claim that the core was able to detect and correct all injected faults. Another solution is described in [28], where a 666 MHz Arm A9 hard core and a 25 MHz LowRISC softcore on Zynq-7000 run in a dual-core lock-step (DCLS) execution. Each core receives the same input, and additional hardware compares the output. The system can stop, restart, restore from a checkpoint, or continue execution. Specific software modules were used to test the processor architecture, the synchronization module, and the checker module, respectively. Full coverage is not ensured, and

TABLE 1. Fault Tolerant RISC-V core summary expanded and updated from [36]; the works are divided by FT techniques, units and verification methodologies.

Work	Core	FT Techniques	FT units	Verification mean
[38]	CEVERO (FT Ibex)	Dual core Lock-step	Entire microarchitecture	Emulation
[39]	SHAKTI-F	NMR and standard ECC	ALU, PC, Reg.File, memory	Simulation & Emulation
[40]	Technolution BV	NMR and Hamming	Entire microarchitecture	Simulation & Emulation
[41]	FT lowRISC	Distributed TMR & parity check	Reg.File	Emulation
[42]	Taiga	Distributed TMR	Configuration memory	Radiation testing
[43]	FT Rocket	Distributed TMR	Entire microarchitecture	Simulation & Emulation
[44]	FT lowRISC	Application-tailored TMR	ALU	Emulation
[36]	Ad hoc core	Partial TMR and Hamming	ALU, PC, Reg.File, Control logic	Simulation
[28]	LowRSSC & Arm	Dual core Lock-step	Entire microarchitecture	Emulation
This work	Klessydra-fT03	Buffered-TMR	Reg.File, Write-Back unit, PC, Load Store Unit	Simulation

so-called common-mode faults may occur, i.e. faults that affect both systems failing the computation.

In [43], the authors performed a deterministic FI campaign to detect the critical bits of two different implementations of the Rocket RISC-V processor: one protected by the distributed TMR technique and one unprotected. The results showed that the unprotected version could already obtain correct results in 96% of the FI cases, whereas the TMR version augmented the ratio to 99.9%. The work in [40] uses N-modular redundancy (NMR) and Hamming codes in architecture and pipeline registers, whereas FI is performed using a dedicated FI block placed in different positions of the processor architecture.

In [41], the authors propose a technique able to protect the register file of the lowRISC processor implemented on a Nexys 4 DDR board featuring a Xilinx Artyx-7 FPGA. By duplicating the entire register file and using a parity-checking mechanism, the authors can reduce the error propagation from 10.17% to 0%. The authors of [42] present the BL-TMR software suite, which can analyze a Xilinx Vivado netlist and add triple redundancy to critical nodes. The solution was applied to the design of the RISC-V Taiga processor implemented on a Kintex Ultrascale FPGA in two different configurations, with TMR and without, and was tested by neutron radiation, with a 24X improvement in mean failure rate at the cost of 5.6X overhead of logic resources. Similarly, the same authors in [45] use the SpyDrNet tools to apply TMR on some RISC-V processors, observing reductions of the failures due to configuration RAM errors between 20X to 500X, while the work reported in [44] exploits the idea that only the most statistically frequent ALU operations require protection to reduce hardware overhead.

In [36], the authors propose a fault-tolerant RISC-V processor core using Hamming code to protect the register file and the program counter, and TMR to protect the ALU and control logic. The single-cycle microarchitecture was implemented on a Xilinx Zynq FPGA, reaching a frequency of 50 MHz. The authors extended the use of the core to a microcontroller system-on-chip (SoC), discussing the architecture, synthesis results, and comparisons with other cores using random FI simulations.

Overall, existing studies on RISC-V processor hardening by design compose a heterogeneous scenario, ranging from solutions dedicated to specific functional units to full TMR protection of the microarchitecture, but none of them covers the exploitation of IMT to implement FT in a RISC-V compliant architecture. Thus, the proposed work represents an interesting alternative to be explored in the RISC-V context.

III. MICROARCHITECTURE DESIGN

A. BASELINE MICROARCHITECTURE

The concept of the proposed work is to quantify the FT performance of the triple redundancy obtained by properly adapting an IMT core that supports three hardware threads. The open-source Klessydra-T [46] is an IMT processor core family based on a four-stage in-order pipeline that interleaves three or more hardware threads (Fig. 1). It is fully compatible with the PULPino open-source microcontroller platform [34] and supports the RV32IMA instruction set in bare-metal execution. It also supports a custom ISA extension "K", designed to accelerate vector computations. The K-extension and related hardware units can be excluded from the synthesis configuration to generate the Klessydra-T03 core.

Our work addresses the resilient execution of the standard RV32IMA instruction set; therefore, we used Klessydra-T03 as the baseline on which we built the FT microarchitecture design to be analyzed, named Klessydra-fT03. Furthermore, although the core is compatible with the PULPino platform, the peripherals included in the platform architecture are not within the scope of this study and allow a fair comparison with previous studies on other FT RISC-V cores.

B. PRINCIPLE OF IMT-BASED FT

Like other FT techniques with SMT and MC multi-threading schemes, the proposed approach exploits the IMT scheme to merge spatial redundancy with temporal redundancy, introducing checking/voting logic and accepting the consequent throughput decrease. The architecture executes three threads that are instances of the same program, maintaining the state of the threads in redundant registers (spatial redundancy) and sharing combinational logic among the threads by interleaving their instructions with one cycle time distance

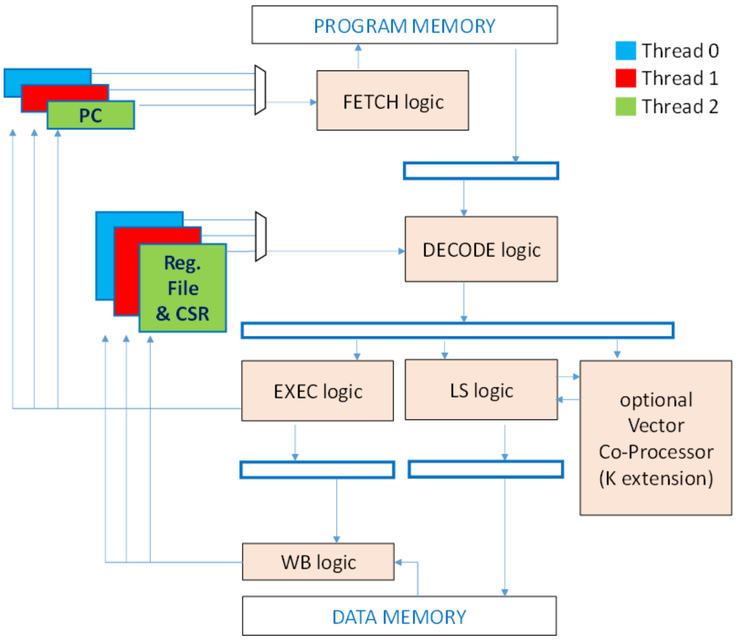


FIGURE 1. **Klessydra-T03 microarchitecture scheme. Colors evidence the sequential elements that maintain the state of each thread separately.**

(temporal redundancy). Each of the three threads has a numeric identifier, and the order of the interleaved instruction fetch is always thread 2, thread 1, and thread 0.

The original IMT architecture natively offers spatial redundancy using a replicated program counter, register file, and control/status registers (CSRs) to maintain the states of the three threads being executed. Voting can be introduced at several points in the pipeline microarchitecture among the logic signals produced by three identical threads. Fig. 2 summarizes the different schemes that can be considered. Each possible choice requires hardware and time overhead to maintain a consistent program flow. In the following section, we discuss and illustrate the design choices referring to the specific architectural elements to be protected.

C. REGISTER DATA FLOW PROTECTION

The scheme shown in Fig. 2(a) performs voting among the data replicas produced by the three threads at the output of the replicated register file. Notably, voting can occur only when the latest of the three threads (Thread 0) reads its operands because otherwise, the three copies of the data may not be ready in the registers. Therefore, if the voting fails, and the faulty operand is the one read by Thread 1 or Thread 2, the pipeline is flushed to avoid incorrect processing values, resulting in a 2-cycle latency overhead. In addition, the scheme is more efficient than the classical TMR implemented at the flip-flop level inside registers because the voting logic only operates on the output ports of the register files.

Scheme (a) cannot prevent a thread from writing in an incorrect destination register because of a faulty bit in the write-back buffer. While this is going to be detected by the voting mechanism when the data will be read by a subsequent instruction, it opens the way for fault amplification (one faulty bit in the write-back buffer generates two wrong 32-bit words, i.e. the register that is not written and the register that is mistakenly overwritten), which increases the probability that a subsequent single-bit fault in the register file will not be detected.

Schemes (b) and (c) implement a more conservative solution by buffering all the signals related to the write-back operation of Thread 2 and Thread 1 into dedicated registers, and then vote among the write-back operations of the three threads when Thread 0 reaches the write-back phase before writing into the register file. We refer to this basic mechanism as a *Buffered TMR*.

Scheme (c) differs from the scheme (b) in that it renounces having a replicated register file maintaining the states of the three threads separately by having a single register file equipped with ECC support. Plausibly, this scheme may have less tolerance to multiple faulty bits in a register, with the advantage of a smaller hardware cost. However, in a single fault event scenario, the vulnerability factor is the same as in scheme (b). In our analysis, all the presented results refer to the more general structure represented by the scheme (b).

Schemes (b) and (c) eliminate the possibility that any thread writes to an incorrect destination register at the expense of an additional hardware overhead. In fact, write-back voting can occur only when Thread 0 reaches the write-back phase; therefore, the register is only written

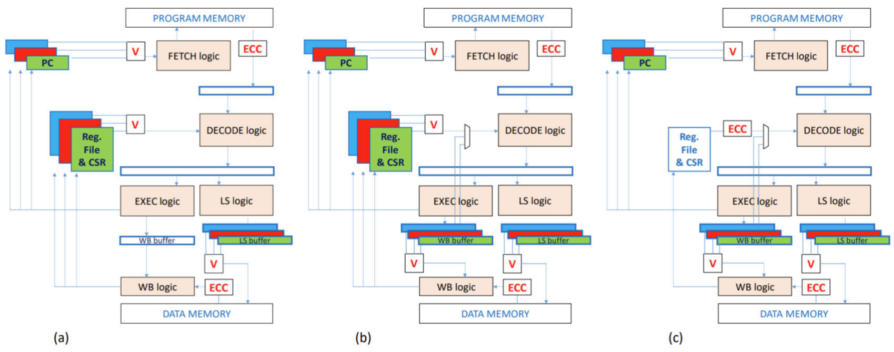


FIGURE 2. *Buffered TMR* **microarchitecture schemes; (a) (b) and (c) share the same protection mechanism for the Load/Store (LS) unit. (a) cannot prevent a thread from writing in an incorrect destination register, (b) and (c) buffer all the signals related to the write-back operation and then vote when Thread 0 reaches the write-back phase. (c) does not have a replicated RF saving in HW cost.**

at that point. A form of bypass logic is needed between the write-back and decode stages to avoid stalling Thread 2 and Thread 1 in the case of read-after-write data dependency, as represented by the multiplexer in Fig. 2(b) and (c). The operation is illustrated by the timing diagram in Fig. 3, in which a specific instruction processing phase called "'*Buffered* Write-back"' is introduced (letter "'B"' in the figure), in addition to the classical Fetch, Decode, Execute, and Write-back.

In case write-back voting detects a fault, dedicated voting logic ensures that the correct value is written back in the correct register. In addition, if the faulty instruction had forwarded an incorrect result value via the bypass logic, the voting logic forces the subsequent instruction of the same thread to read the corrected value again, resulting in the repetition of the decode phase with one clock cycle penalty in the pipeline. The timing of this mechanism is illustrated in Fig. 4, where a specific instruction phase called "'Halt"' is introduced to represent the stall of the pipeline while waiting for the correct value to be read.

D. MEMORY DATA FLOW PROTECTION

Schemes (a), (b), and (c) share the same protection mechanism related to the Load/Store (LS) unit. In contrast to a normal multi-threading operation, the status of the three threads is exactly replicated, so they share the same stack region as well as static variables. Therefore, it is necessary to implement a load/store unit that avoids performing replicated load/store accesses to the same location, resulting in malfunctions when reading from memory-mapped I/O peripherals or inconsistent behaviour, respectively. In addition, replicated access results in energy wastage. However, the LS unit must avoid propagating faults to and from the memory subsystem.

The modified LS unit applies the *Buffered TMR* mechanism to memory operations, buffering the memory access signals of Thread 2 and Thread 1 and performing only Thread 0 memory accesses. When Thread 0 executes the memory operation, the buffered signals are voted on before accessing the memory. Furthermore, to tolerate faults occurring in memory cells, data loaded from memory are ECC-protected so that the in-memory SEU cannot affect the correctness of a load operation.

E. PROGRAM CONTROL FLOW PROTECTION

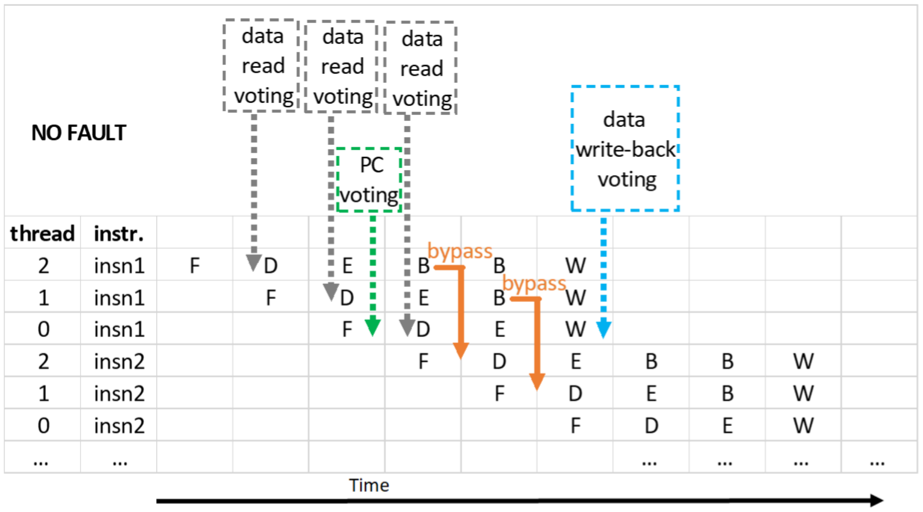
All three schemes in Fig. 2 vote on the program counter (PC) to avoid a fault causing an unwanted jump in the program execution flow. PC voting can occur only when Thread 0 fetches a new instruction; otherwise, the three PC copies are not yet ready. In the case of a PC fault, the instruction fetch is repeated using the correct PC value. The order of thread interleaving must not change. Therefore, depending on which of the three threads results in an incorrect PC value, the fetch recovery operation may imply an overhead of up to three clock cycles. Fig. 5 illustrates the case of a PC fault on thread 0, resulting in a one-cycle delay in the pipeline operation.

IV. PERFORMANCE EVALUATION ENVIRONMENT AND METHODOLOGY

Defining a reproducible FI simulation paradigm and figure of merit is essential for correctly evaluating the augmented resilience achieved by fault-tolerant microarchitecture designs. We evaluated the FT performance of the Klessydra-fT03 core with respect to the original multithreading T03 core and single-thread Klessydra-S1 core using an FI methodology aimed at evaluating and comparing the architecture vulnerability factor (AVF) improvement. Furthermore, we compared fT03 with the published results obtained by existing RISC-V FT cores exposed to random FI by reproducing the same testing environments [36] and highlighting the differences in terms of performance, hardware cost, and fault resilience. All the FI simulations used in this work are cycle-accurate.

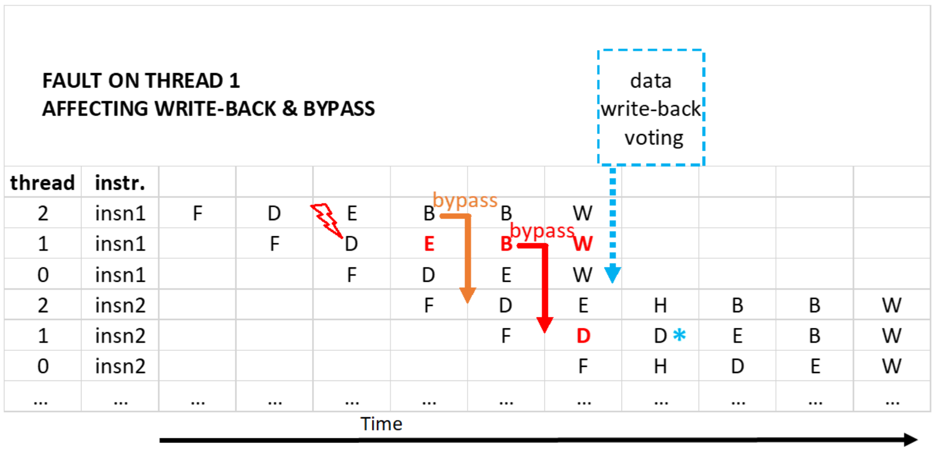
A. BASIC METHOD AND DEFINITIONS

In an RTL description, signals assigned to synchronous processes are translated into synchronous registers in a synthesized netlist. Thus, injecting bit flips into such signals



thread	instr.										
2	insn1	F	D	E	B	B	W				
1	insn1		F	D	E	B	W				
0	insn1			F	D	E	W				
2	insn2				F	D	E	B	B	W	
1	insn2					F	D	E	B	W	
0	insn2						F	D	E	W	
...	...							...	...	...	...

FIGURE 3. Timing diagram of Buffered TMR architecture in absence of faults: Fetch (F), Decode (D), Execute (E), Buffered Write-back phase (B). Bypass is performed only in case of data dependency, data write-back voting is done at the end of Thread 0 write-back, while PC voting is done when Thread 0 fetches the new instruction.



thread	instr.										
2	insn1	F	D	E	B	B	W				
1	insn1		F	D	E	B	W				
0	insn1			F	D	E	W				
2	insn2				F	D	E	H	B	B	W
1	insn2					F	D	D *	E	B	W
0	insn2						F	H	D	E	W
...	...	...	...	...	...	...	...	...	...	...	...

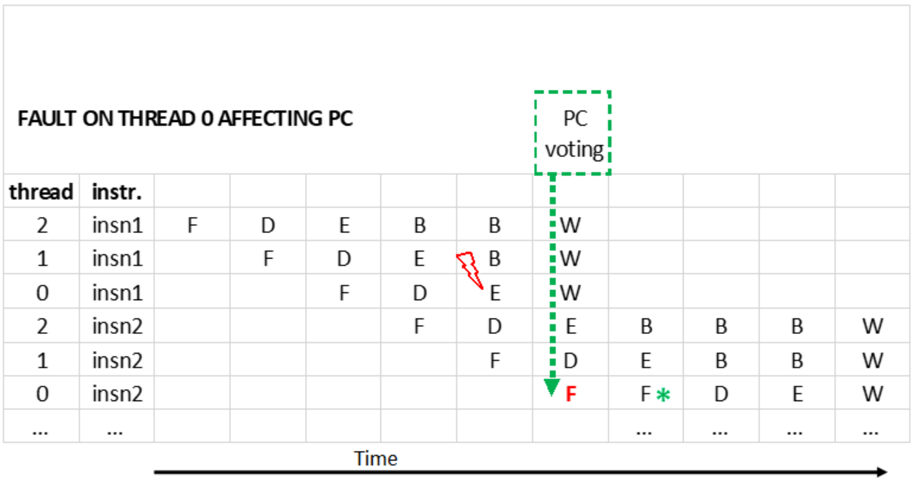
FIGURE 4. Recovery mechanism in case of data read fault in Thread 1. A fault hits the decode stage of Thread 1. When data write-back voting is performed, a halt step (H) starts with the subsequent correction (D*). Faulty signals/data are represented by red color.

during the RTL simulation allows us to observe the effect of SEUs hitting synchronous registers in the microarchitecture under analysis.

The FI campaign was implemented in a universal verification methodology (UVM) mixed-language environment running on the QuestaSim simulator. First, a complete list of the synchronized signals is generated by a parsing algorithm of the RTL VHDL description of the processor. Then, a SystemVerilog testbench reads the list of the target signals and automatically runs the FI simulation of the VHDL code. The simulation covers the entire execution of an application program by the simulated processor. Each FI experiment can simulate the injection of a value flip onto any target bit of the processor, in one or more specific time instants selected within the runtime of the program execution simulation.

The adopted FI approach targets every synchronous signal of the RTL description of the processor, that will result in flip-flops in the hardware synthesis, with bit-level accuracy. Thus, the simulation goes beyond the Instruction Set Architecture (ISA) resources and is capable of injecting faults into all the internal registers of the microarchitecture.

The functionally correct operation of the FI environment was verified by tests based on the official RISC-V random instruction generator with different random seeds to maximize the functional coverage of the RTL design.



thread	instr.										
2	insn1	F	D	E	B	B	W				
1	insn1		F	D	E	B	W				
0	insn1			F	D	E	W				
2	insn2				F	D	E	B	B	B	W
1	insn2					F	D	E	B	B	W
0	insn2						F	F*	D	E	W
...	...							...	...	...	...

FIGURE 5. Timing diagram of the operation in case of PC fault. A fault hits the execute stage of Thread 0, resulting in an incorrect fetch step. PC voting is performed and the correct value is re-fetched at the next cycle. The faulty fetch (F) is represented by red color, followed by the corrected fetch (F*).

The environment can identify five different error categories occurring during the simulation, classified according to the outcome of the simulated execution:

- Correct outcome: The execution successfully ends with the same content in the data memory as in the error-free execution.
- Incorrect termination: Program execution terminates prematurely.
- Incorrect data: Data inconsistency: The program execution terminates normally, yet the data memory content is inconsistent with the error-free execution case (also known as silent data corruption).
- Crash with hang: execution never ends (infinite loop).
- Crash with exception: execution results in an unrecoverable processor exception.

For this study, we grouped the different outcome types into two categories: success and failure, tracing all incorrect and crash outcomes as failures. In the FI analysis, we partially took into account latent errors. In fact, we assume that in the target application domain – bare metal embedded systems – the core repeatedly executes the same program and is reset between consecutive program runs. Regarding memory content errors, at the end of the simulated program execution, we compared the entire memory image with the one produced by the correct program execution. A failure was reported in case of a mismatch. As a result, any potential latent error in the memory that may affect the next program execution was taken into account in our analysis.

In order to quantify the vulnerability of the architectures under analysis, we refer to the widely adopted metric of the architectural vulnerability factor (AVF) defined as:

$$\mathrm{AVF} = \frac{N_{ACE}}{N} \tag{1}$$

where N is the total number of bits in the architecture and N_{ACE} is the average number of processor bits that influence the architecturally correct execution (ACE) of the program in a clock cycle [11], [12], [47]. By definition, a bit of the hardware microarchitecture is ACE in a certain clock cycle if a value change occurring on that bit in that clock cycle causes a program failure. An unprotected bit of the architecture passes from ACE to non-ACE state and vice-versa in a deterministic way, depending on the program flow. Since physical events causing SEUs may occur at random instants during the program execution, a generic bit j has a certain probability $P_F(j)$ of being in ACE state when a SEU occurs. The average number of bits in ACE state in the whole microarchitecture when a SEU occurs, N_{ACE}, can be expressed as

$$N_{ACE} = \sum_{j=1}^{N} P_F(j) \tag{2}$$

In addition to the AVF parameter, we also consider the results in terms of the Mean Work to Failure (MWTF) metric, which expresses the average amount of work that an application can perform until reaching a failure [48], [49], thus introducing the impact of program duration. The MWTF can be obtained as

$$\mathrm{MWTF} = \frac{1}{Rf \cdot \mathrm{AVF} \cdot T\_exec} \tag{3}$$

where Rf is the raw fault rate and T_{exec} is the application program execution time. Since the purpose of our analysis is the relative comparison of microarchitectures, we are interested in the MWTF normalized with respect to the fault rate.

B. TIME-FRAME-SPAN METHODOLOGY

The FI approach adopted to evaluate the proposed microarchitecture scheme is similar to that reported in [50]. In contrast to injecting randomly distributed faults during the program execution (Monte Carlo statistical approach), it employs the deterministic injection of faults on each target bit of the architecture in specific time intervals, within the program execution duration, in order to assess the average number of ACE bits in a cycle.

We divide the total execution time into m intervals called time frames and we run the RTL simulation of the processor executing the whole benchmark program, injecting a fault on a target bit j of the processor for the duration of a time frame and repeating the analysis for all the time frames. In general, fault-injection on bit j reports a system failure when faults are injected only in m_F time frames out of the total m time frames. Assuming that SEU physical events in the real system occur with a uniform time distribution with respect to the program execution duration, we can estimate that the probability for bit j of being in ACE state when a SEU occurs is

$$P_F(j) \leq \frac{m_F(j)}{m} \tag{4}$$

which can be used to estimate an upper bound of N_{ACE} according to (2). Unlike a statistical Monte-Carlo approach, the analysis deterministically identifies the time frames in which the system is immune to faults on bit j during program execution. Each time frame duration was calculated as 1/10 of the benchmark execution, in order to estimate the probability for the target bit of being in ACE state (expressed in percentage) with 10% steps. For instance, if only 8 out of 10 time-frame simulations result in a program failure, we can assume that bit j is ACE for at most 80% of the execution time, for the given benchmark. In our experiments, we assumed a clock cycle of 10 ns and an FI rate of one fault every 15 cycles on the target bit of the simulation run. This scheme leads to 6667 faults injected into each bit of the microarchitecture every 1 ms of program execution. To the best of our knowledge, this is the highest density of faults ever analyzed in a fault-injection simulation campaign reported in the literature. In order to ensure the correct program termination, the simulation waits for a hundred clock cycles after the end of the fault-injection in the last time frame of the simulated execution.

C. BENCHMARK PROGRAMS

To evaluate the resilience of the proposed architecture and the reference ones in executing real application kernels, we referred to eight benchmarks, spanning from typical AI kernels to cryptography kernels and signal processing kernels, namely:

- convolution2d: 2D image convolution.
- crc32: 32-bit cyclic redundancy check decoding.
- fft: fast Fourier transform calculation.
- fir: Finite impulse response filter.
- aes-cbc: Advanced encryption standard cypher-block chaining decryption algorithm.
- sha: Secure hash algorithm calculation.
- ipm: Inflexion point method calculation.
- fdctfst: Fast discrete cosine transform calculation.

We assumed bare-metal execution of the benchmarks, with no operating system kernel running on the cores. For each benchmark, the total execution time was divided into 10-time frames. Table 2 details the execution time and bit fault count per time frame for each benchmark, for the three cores, according to the adopted FI rate of 1 fault every 15 cycles. While the fT03 core executes three identical interleaved replicas of the benchmark program to implement the RMT FT, the S1 and T03 cores execute only one thread running the benchmark program. In particular, since T03 is an IMT architecture interleaving three threads; we tested it by interleaving the benchmark thread with two idle threads, without including the idle thread failures in the failure count, *unless they also caused a failure in the benchmark thread execution*, for a fair comparison.

V. ARCHITECTURE VULNERABILITY ASSESSMENT

Several interesting results emerged from the FI campaigns for cores S1, T03, and fT03. Fig. 6 shows the breakdown of the system failure probabilities associated with different units of the hardware microarchitectures if a fault hits the unit during the execution of the benchmark program. The units selected were those that were the most vulnerable to analysis. While the tests addressed all the individual bits of the hardware microarchitectures, arithmetic averages per unit were plotted for simplicity.

For all analyzed benchmarks, there was a significant reduction in failures associated with all hardware units in the fT03 fault-tolerant microarchitecture with respect to the non-hardened S1 and T03 microarchitectures. KlessydraT03 resulted in the possibility that a fault, hitting a unit executing one of the idle threads, indirectly causes a fault in the benchmark execution thread; thus, S1 is statistically less vulnerable.

Comparing Fig. 6 with Table 2, the total cycle count seems to affect the AVF in the sense that a longer benchmarks exhibit higher AVF. This can be ascribed to the fact that a long benchmark is likely to use (and thus expose to failure) more hardware registers than a short benchmark.

The FI simulation produced a large quantity of data that could be used to calculate the AVF associated with the execution of each benchmark for each of the three cores. The AVF results are presented in Table 3. Because lower AVF values reflect high resilience; it was possible to observe that in some of the benchmarks, the resilience of the T03 core is improved by one order of magnitude in the fT03 core.

A similar performance can be observed from MWTF metric. Table 4 reports the MWTF normalized with respect to the fault rate, for the three cores under comparison. In the table, where higher values represent higher resilience, it is possible to observe that the fT03 core still maintains an order of magnitude advantage over other architectures.

It is interesting to observe the hardware cost of such a significant improvement. Table 5 lists the resource utilization of the three microarchitectures synthesized by Xilinx

TABLE 2. Benchmark setup for fT03, T03 and S1 architectures. The total execution time is reported in [ms] with a 10-frame division and a fault every 15 clock cycles under a specific amount of faults per frame.

	fT03		T03		S1	
Bench-mark	**Execution Time** [ms]	**Faults per frame**	**Execution Time** [ms]	**Faults per frame**	**Execution Time** [ms]	**Faults per frame**
fft	1,595	907	1,342	912	0,645	369
fir	0,725	414	0,646	422	0,321	181
aes	1,385	790	1,244	787	0,544	311
ipm	0,204	116	0,178	121	0,095	54
fdctfst	0,098	55	0,090	63	0,053	30
sha	1,792	1022	1,589	1024	0,680	390
crc32	0,200	113	0,185	140	0,115	65
conv2	0,257	144	0,216	146	0,106	60

TABLE 3. AVF estimation from (1) for all the analyzed benchmarks, it is possible to see the architectural vulnerability factor improved by an order of magnitude with respect to T03 and S1, reflecting the high resilience and low vulnerability of fT03.

Benchmarks	**T03**	**S1**	**fT03**
fft	$0,5257$	$0,4251$	$0,0654$
fir	$0,5005$	$0,3881$	$0,0512$
Aes	$0,5409$	$0,4390$	$0,0762$
Ipm	$0,4354$	$0,3425$	$0,0426$
fdctfst	$0,3640$	$0,2345$	$0,0292$
sha	$0,5372$	$0,4362$	$0,0762$
crc32	$0,3396$	$0,2524$	$0,0179$
conv2	$0,4712$	$0,3604$	$0,0532$

TABLE 4. By applying (4), it is possible to evaluate the mean work to failure estimation for the analyzed benchmarks under a specific radiation rate, considering the trade-off between reliability and performance. Using the AVF values from Table 3 and Execution time values from Table 2, MWTF still confirm high resilience for the fT03 core.

Benchmarks	**T03**	**S1**	**fT03**
fft	$1,42E+03$	$3,65E+03$	$9,59E+03$
fir	$3,09E+03$	$8,03E+03$	$2,69E+04$
Aes	$1,49E+03$	$4,19E+03$	$9,47E+03$
Ipm	$1,29E+04$	$3,06E+04$	$1,15E+05$
fdctfst	$3,05E+04$	$8,03E+04$	$3,50E+05$
sha	$1,17E+03$	$3,37E+03$	$7,32E+03$
crc32	$1,59E+04$	$3,43E+04$	$2,79E+05$
conv2	$9,81E+03$	$2,62E+04$	$7,31E+04$

TABLE 5. Synthesis results on a Xilinx Kintex-7 FPGA, showing that fT03 consumes more energy than T03 and S1.

Core	**LUTs**	**FFs**	**Energy [pJ/cycle]**
S1 (non-hardened)	3528	1959	340
T03 (non-hardened)	5527	4489	380
fT03 (hardened)	6670	4910	390

Vivado 2019 on a Kintex-7 FPGA. Klessydra-fT03 can reach 197 MHz, whereas Klessydra-T03 with Klessydra-S1 can be successfully synthesized at a target frequency of 220 MHz.

Regarding energy consumption, Table 5 shows that Klessydra-fT03 consumes slightly more energy than its non-hardened companion T03, while the single-threaded Klessydra-S1 core does not consume 1/3 of the energy consumed by the three-threaded cores owing to the efficiency of the IMT microarchitecture scheme.

VI. COMPARISON WITH EXISTING FAULT-TOLERANT RISC-V DESIGN CASE STUDIES

Generally, it is difficult to directly compare different FT cores unless they are subjected to the same FI test. Nonetheless, we attempted to systematically analyze and discuss the state-of-the-art RISC-V FT results in the literature.

Existing resilience analysis studies on FT processor cores differentiate between the type of FI technique applied and the type of results reported. Table 6 lists the FT RISCV processors presented in Section II, along with the corresponding analysis characteristics extrapolated from the literature.

The works in [28], [38], [39], and [40] use an FI hardware block to emulate the fault events in the synthesized architecture. This approach has the advantage of being executed on the target FPGA; however, it is strictly customized to the target design. In [42], a neutron radiation process is performed. The actual number of injected bit upsets is related to the chosen radiation fluence (e.g. 2.00 x 10^{11} n/cm2), yet it is difficult to quantify. The works in [43], [44], and [41] use FPGA built-in Ips, along with related software tools, capable of inserting faulty bits into the configuration memory of SRAM-based FPGAs. Targeting the FPGA configuration memory addresses the eventuality of SEU-induced modifications in the microarchitecture structure and in the content of LUTRAMs; however, such techniques lack the possibility of injecting faults in the logic values contained in the architectural registers of the design implemented by FFs on the FPGA. According to [43], 95% of the faults injected in an FPGA configuration SRAM do not result in any failure of the processor's operation in the absence of any FT countermeasure. Moreover, state-of-the-art FPGAs have advanced built-in features for runtime error detection and correction in configuration SRAM [51]. However, they cannot mitigate register content corruption induced by SEUs. Therefore, in this study, we addressed the FI of logic values in the microarchitecture registers.

In [36] and in the proposed work, FI is performed by a dedicated simulation environment. In [41] and [44], only specific microarchitecture blocks are targeted, i.e., the register file and the integer ALU, respectively, rather than the whole processor. Similarly, in [39], fault events are emulated only in specific units of the microarchitecture.

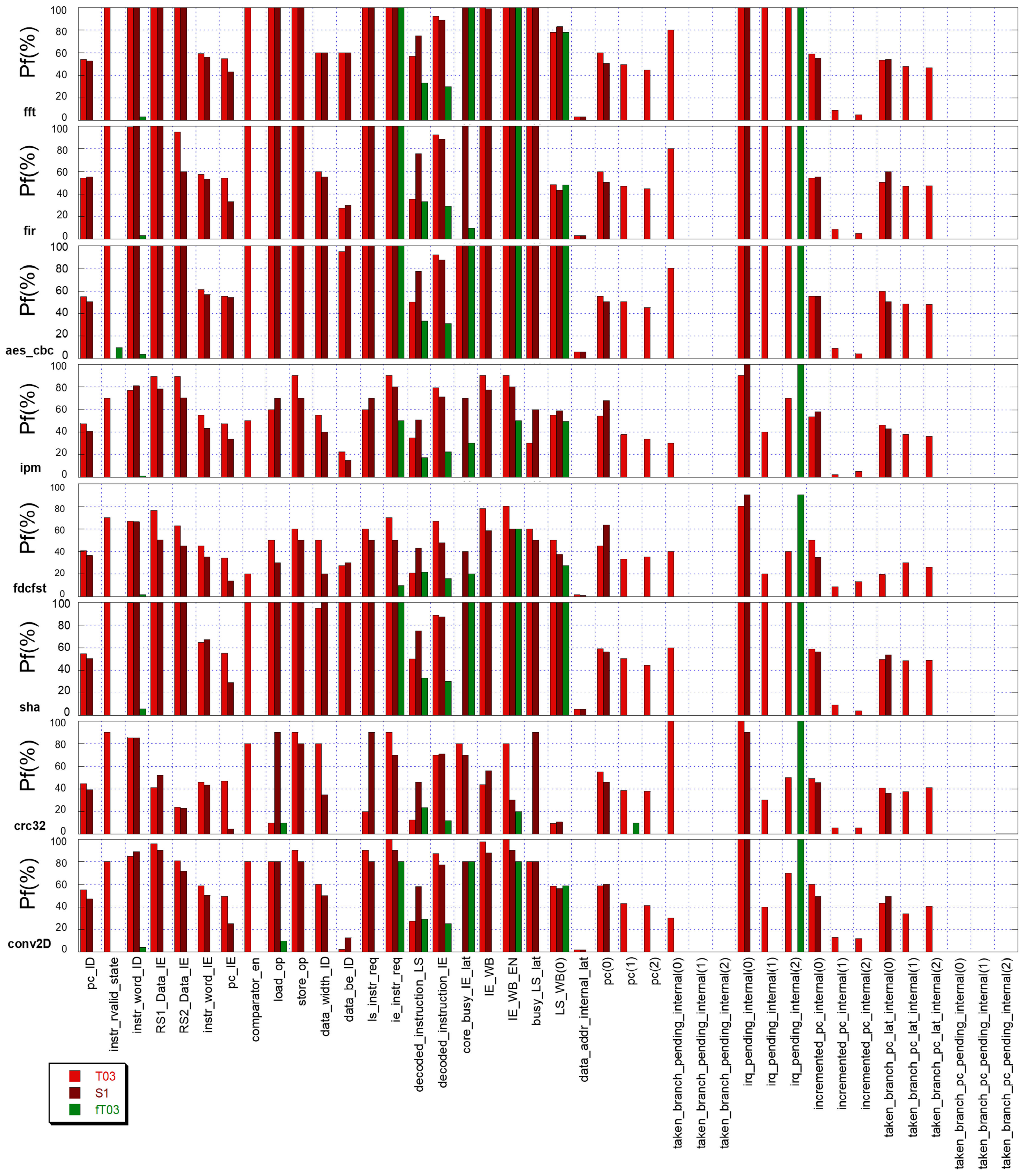


FIGURE 6. Results for fft, fir, aes, ipm, fdctfst, sha, crc3, conv2D FI tests, obtained following the setups showed in Table 2. For each benchmark, a 10-time frames simulation is launched, injecting for each bit of the architecture the amount of faults reported in Table 2.

The works in [28], [38], [39], [40], and [44] do not report the quantity of injected faults, whereas [36] reports a low quantity of them with respect to the remaining works. In the proposed study, we injected up to 8.570.000 faults into the core microarchitecture, targeting the ACE bits on the longest benchmark.

TABLE 6. Fault-injection results comparison for the RISC-V cores listed in Table 1. (NFP = Normalized Failure Percentage) Annotations: (1) Authors do not report failure rate of the unprotected architecture (2) FI affecting some specific units (3) FI affecting only the protected units (4) FI affecting only the instruction register.

Work	fault-injection method	Benchmarks	Hardware resource overhead	Performance overhead	Fault recovery time overhead	Reported Fault Tolerance result
[38]	FI hardware block	Fibonacci	>2X FFs, >2X LUTs	Not declared	40 cycles	Zero failures (1)(4)
[39]	FI hardware block	Instruction Generator	1.39X FF 1.17X comb	1.54X cycle time	<4 cycles	Zero failures (1)(2)
[40]	FI hardware block	Dhrystone, Coremark	3.70X FF 5.96X LUT	3.3X cycle time	zero cycles	Zero failures (1)
[41]	FI on FPGA config. mem	Quicksort, FFT, MatMul	2.46X LUT	zero	zero cycles	Zero failures (1)(3)
[42]	Neutron radiation	Dhrystone	3.00X FF 5.64X LUT	1.4X cycle time	zero cycles	4.1% NFP
[43]	FI on FPGA config mem	Dhrystone, Systest, Linpack, Whetstone	>3X FFs, >3X LUTs	Not declared	zero cycles	2.3% NFP
[44]	FI on FPGA config mem	Quicksort, MatMul, Dijkstra, Fibonacci	from 1.14X LUT to 2.94X LUT	Not declared	zero cycles	from 40% NFP to 5.72% NFP (3)
[36]	FI simulation	Coremark, VectSum, CCSDS-123	1.19X FF 1.70X LUT	1.5X cycle time	zero cycles	7.7% NFP
[28]	Only functional test of the error detection module	Ad-hoc benchmark	1.04X FF 1.01X LUT + 1 ARM core	Not declared	Software dependent (not reported)	Not available
This work	FI simulation	FFT, Fir, Sha, Aes-Cbc, Crc32, Conv2D, Ipm, Fdctfst	1.09X FF 1.17X LUT	3X cycle count	<3 cycles	2.9% NFP

The authors of [38], [39], [40], and [41] declare that their architectures are resilient to all the injected faults. However, [38] and [41] refer to FI concentrated on protected units only, and [39] refer to FI concentrated on specific units rather than on the entire microarchitecture. On the other hand, the design reported in [40] exhibits the highest overhead in terms of both time and hardware resources. The works in [43], [44], and [36] present the error percentage for each benchmark and use a random FI on the entire architecture, allowing us to perform a comparison, although the confidence level of the results is limited by the differences in the number of randomly injected faults and the statistical error margin [50]. As introduced in [43], to compare two different FT design approaches; it is possible to refer to the normalized failure percentage (NFP), obtained by dividing the failure percentage of the protected microarchitecture by the failure percentage obtained on the unprotected version of the same microarchitecture subjected to the same FI for all the designs to be compared. Table 6 lists the NFP for similar benchmarks from [36], [42], [43], and [44], and Klessydra-fT03. In [42], the authors declare a 24X improvement in the average time between failures; thus, we can estimate an NFP of $1/24 = 4.1\%$ by assuming a uniform time distribution of radiation-induced faults.

Klessydra-fT03 exhibits a low NFP as in [43], whereas [43] obtains its resilience at the expense of 3X hardware overhead (with undeclared performance overhead), the resilience of Klessydra-fT03, based on *Buffered TMR* in IMT execution,

TABLE 7. Fault-injection comparisons after 10 SEUs with respect to [36]. Results show the failure rate on both *sum of vectors* and *Coremark* benchmarks. The non-Hardened T03 core is running with one active and two idle threads (due to the IMT architecture).

Core	Sum of vectors	Coremark
[36] - Non-Hardened core	99%	100%
[36] - Hardened core	35%	6%
Klessydra-T03 non-hardened core	51%	27%
Klessydra-fT03 hardened core	4%	2%

TABLE 8. Clock frequency comparison of fault-tolerant RISC-V cores.

Work	Core	Implementation	Clock Frequency [MHz]
[38]	CEVERO (FT Ibex)	FPGA	not avail.
[39]	SHAKTI-F	ASIC	330,00
[40]	Technolution BV	FPGA	36,10
[41]	FT lowRISC	FPGA	100,00
[42]	Taiga	ASIC	165,00
[43]	FT Rocket	ASIC	not avail.
[44]	FT lowRISC	FPGA	100,00
[36]	Ad-hoc core	FPGA	49,99
[28]	LowRISC & Arm	FPGA (LowRISC core)	25,00
This work	Klessydra-fT03	FPGA	197,00

is obtained at the expense of 3X time overhead. An advantage of the Klessydra-fT03 approach is that the fault-tolerant operating mode can be disabled at any time, thereby achieving full utilization of the hardware for multi-threading performance when executing non-critical applications or during periods of non-harsh operating conditions.

A particular interesting test case is detailed in [36], which allows a direct comparison with the proposed method. In [36], the authors performed 100 different runs by injecting 10 random faults into randomly chosen microarchitecture bits during each execution run. We refer to this experiment as the 100 x 10 SEU FI simulation. We replicated the 100 x 10 SEU tests running the same benchmarks as in [36] on Klessydra-fT03 and -T03. Although a fair comparison using a random FI is inherently affected by the random differences occurring in the experiments, we summarize a set of interesting results in Table 7, showing the percentage of runs that ended with a failure. Klessydra-fT03 presents a very low error percentage for both benchmarks. Interestingly, non-hardened Klessydra-T03 also exhibits relatively low failure percentages. This feature can be explained by the fact that to be aligned with the operation of the other cores, Klessydra-T03 runs one active thread and two idle threads in the IMT pipeline so that errors hitting the state of idle threads do not produce visible effects.

To complete the comparison with the speed performance results, Table 8 presents the clock frequencies achieved by different fault-tolerant processors. Klessydra-fT03 reaches the highest value among the FPGA implementations.

VII. DISCUSSION

Both the AVF and the MWTF results show that the Buffered TMR technique improves resilience by one order of magnitude over the corresponding non-hardened microarchitecture. Furthermore, the NFP results show that the resilience achieved by the Buffered TMR approach is comparable to protecting a single-threaded core with full TMR, at the cost of execution time overhead instead of hardware overhead.

Like any TMR-based approach, the proposed technique is characterized by non-negligible overhead (time overhead for the proposed technique), and as such it is well-suited to systems that are exposed to a relatively high fault rate or very critical applications, at least for part of their operation time. Yet, a significant feature of the IMT-based approach is the possibility of switching to normal multi-thread execution, thus offering flexibility of use when the operating conditions or the application run do not require sacrificing performance for resilience. In addition, the IMT microarchitecture intrinsically reaches a higher clock frequency than other solutions, when implemented on the same FPGA device. The power consumption results exhibit a small overhead of the Buffered TMR with respect to the corresponding non-hardened IMT core.

VIII. CONCLUSION

The presented study demonstrates the possible exploitation of IMT processor microarchitectures for FT. The results produced by an extensive FI campaign show that the achievable resilience is comparable to a full TMR, at the cost of time overhead instead of hardware overhead. Yet, the IMT-based approach demonstrates interesting perspectives in terms of flexibility and high clock frequency.

Prospective investigations address multi-bit upsets (MBU) and faults in combinational logic, as well as IMT-based FT in hardware acceleration units.

REFERENCES

[1] Z. Abbas, M. Olivieri, U. Khalid, A. Ripp, and M. Pronath, "Optimal NBTI degradation and PVT variation resistant device sizing in a full adder cell," in *Proc. 4th Int. Conf. Rel., Infocom Technol. Optim. (ICRITO)*, Sep. 2015, pp. 1–6.

[2] U. Khalid, A. Mastrandrea, and M. Olivieri, "Novel approaches to quantify failure probability due to process variations in nano-scale CMOS logic," in *Proc. 29th Int. Conf. Microelectron. (MIEL)*, May 2014, pp. 371–374.

[3] S. Azimi and L. Sterpone, "Digital design techniques for dependable high performance computing," in *Proc. IEEE Int. Test Conf. (ITC)*, Nov. 2020, pp. 1–10.

[4] M. Baleani, A. Ferrari, L. Mangeruca, A. Sangiovanni-Vincentelli, M. Peri, and S. Pezzini, "Fault-tolerant platforms for automotive safety-critical applications," in *Proc. Int. Conf. Compil., Archit. Synth. Embedded Syst. (CASES)*, 2003, pp. 170–177.

[5] L. Blasi, A. Mastrandrea, F. Menichelli, and M. Olivieri, "A space-rated soft IP-core compatible with the PIC® hardware architecture and instruction set," *Adv. Astronaut. Sci.*, vol. 163, no. 2018, pp. 581–594, 2018.

[6] L. Blasi, F. Vigli, A. Cheikh, A. Mastrandrea, F. Menichelli, and M. Olivieri, "An FPGA-based RISC-V computer architecture orbital laboratory on a pocketcube satellite," in *Proc. 5th IAA Conf. Univ. Satell. Missions CubeSat Workshop*, 2019.

[7] C. De Sio, S. Azimi, L. Bozzoli, B. Du, and L. Sterpone, "Radiation-induced single event transient effects during the reconfiguration process of SRAM-based FPGAs," *Microelectron. Rel.*, vols. 100–101, Sep. 2019, Art. no. 113342.

[8] C. De Sio, S. Azimi, L. Sterpone, and B. Du, "Analyzing radiation-induced transient errors on SRAM-based FPGAs by propagation of broadening effect," *IEEE Access*, vol. 7, pp. 140182–140189, 2019.

[9] C. M. Krishna, "Fault-tolerant scheduling in homogeneous real-time systems," *ACM Comput. Surveys*, vol. 46, no. 4, pp. 1–34, Apr. 2014.

[10] R. Devaraj and A. Sarkar, "Resource-optimal fault-tolerant scheduler design for task graphs using supervisory control," *IEEE Trans. Ind. Informat.*, vol. 17, no. 11, pp. 7325–7337, Nov. 2021.

[11] L. Osinski, T. Langer, and J. Mottok, "A survey of fault tolerance approaches on different architecture levels," in *Proc. 30th Int. Conf. Archit. Comput. Syst.*, 2017, pp. 1–9.

[12] I. Oz and S. Arslan, "A survey on multithreading alternatives for soft error fault tolerance," *ACM Comput. Surveys*, vol. 52, no. 2, pp. 1–38, Mar. 2020.

[13] A. Cheikh, G. Cerutti, A. Mastrandrea, F. Menichelli, and M. Olivieri, "The microarchitecture of a multi-threaded RISC-V compliant processing core family for IoT end-nodes," in *Proc. Int. Conf. Appl. Electron. Pervading Ind., Environ. Soc.* Cham, Switzerland: Springer, 2017, pp. 89–97.

[14] T. Strauch, "Acceleration techniques for system-hyper-pipelined soft-processors on FPGAs," in *Proc. Euromicro Conf. Digit. Syst. Design (DSD)*, Aug. 2017, pp. 119–128.

[15] M. Olivieri, A. Cheikh, G. Cerutti, A. Mastrandrea, and F. Menichelli, "Investigation on the optimal pipeline organization in RISC-V multi-threaded soft processor cores," in *Proc. New Gener. CAS (NGCAS)*, Sep. 2017, pp. 45–48.

[16] A. Cheikh, S. Sordillo, A. Mastrandrea, F. Menichelli, G. Scotti, and M. Olivieri, "Klessydra-T: Designing vector coprocessors for multi-threaded edge-computing cores," *IEEE Micro*, vol. 41, no. 2, pp. 64–71, Mar. 2021.

[17] M. Barbirotta, A. Cheikh, A. Mastrandrea, F. Menichelli, F. Vigli, and M. Olivieri, "A fault tolerant soft-core obtained from an interleaved-multi- threading RISC-V microprocessor design," in *Proc. IEEE Int. Symp. Defect Fault Tolerance VLSI Nanotechnol. Syst. (DFT)*, Oct. 2021, pp. 1–4.

[18] Y. Ma and H. Zhou, "Efficient transient-fault tolerance for multithreaded processors using dual-thread execution," in *Proc. Int. Conf. Comput. Design*, Oct. 2006, pp. 120–126.

[19] E. Schuchman and T. N. Vijaykumar, "BlackJack: Hard error detection with redundant threads on SMT," in *Proc. 37th Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw. (DSN)*, Jun. 2007, pp. 327–337.

[20] K. Sundaramoorthy, Z. Purser, and E. Rotenberg, "Slipstream processors: Improving both performance and fault tolerance," *ACM SIGPLAN Notices*, vol. 35, no. 11, pp. 257–268, Nov. 2000.

[21] A. Parashar, A. Sivasubramaniam, and S. Gurumurthi, "SlicK: Slice-based locality exploitation for efficient redundant multithreading," *ACM SIGOPS Operating Syst. Rev.*, vol. 41, no. 11, pp. 95–105, Nov. 2006.

[22] M. A. Gomaa and T. N. Vijaykumar, "Opportunistic transient-fault detection," in *Proc. 32nd Int. Symp. Comput. Archit. (ISCA)*, 2005, pp. 172–183.

[23] M. Peña-Fernández, A. Serrano-Cases, A. Lindoso, M. García-Valderas, L. Entrena, A. Martínez-Álvarez, and S. Cuenca-Asensi, "Dual-core lockstep enhanced with redundant multithread support and control-flow error detection," *Microelectron. Rel.*, vols. 100–101, Sep. 2019, Art. no. 113447.

[24] M. Gomaa, C. Scarbrough, T. N. Vijaykumar, and I. Pomeranz, "Transient-fault recovery for chip multiprocessors," in *Proc. 30th Annu. Int. Symp. Comput. Archit. (SCA)*, 2003, pp. 98–109.

[25] J. Smolens, B. Gold, B. Falsafi, and J. Hoe, "Reunion: Complexity-effective multicore redundancy," in *Proc. 39th Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, Dec. 2006, pp. 223–234.

[26] V. Vargas, P. Ramos, J.-F. Méhaut, and R. Velazco, "NMR-MPar: A fault-tolerance approach for multi-core and many-core processors," *Appl. Sci.*, vol. 8, no. 3, p. 465, Mar. 2018.

[27] S. A. Shernta and A. A. Tamtum, "Using triple modular redundant (TMR) technique in critical systems operation," in *Proc. 1st Conf. Eng. Sci. Technol.*, Nov. 2018, p. 53.

[28] C. Rodrigues, I. Marques, S. Pinto, T. Gomes, and A. Tavares, "Towards a heterogeneous fault-tolerance architecture based on arm and RISC-V processors," in *Proc. IECON 5th Annu. Conf. IEEE Ind. Electron. Soc.*, Oct. 2019, pp. 3112–3117.

[29] G. A. Reis, J. Chang, N. Vachharajani, R. Rangan, and D. I. August, "SWIFT: Software implemented fault tolerance," in *Proc. Int. Symp. Code Gener. Optim.*, 2005, pp. 243–254.

[30] G. A. Reis, J. Chang, and D. I. August, "Automatic instruction-level software-only recovery," *IEEE Micro*, vol. 27, no. 1, pp. 36–47, Jan. 2007.

[31] C. De Sio, S. Azimi, A. Portaluri, and L. Sterpone, "SEU evaluation of hardened-by-replication software in RISC-V soft processor," in *Proc. IEEE Int. Symp. Defect Fault Tolerance VLSI Nanotechnol. Syst. (DFT)*, Oct. 2021, pp. 1–6.

[32] A. Serrano-Cases, F. Restrepo-Calle, S. Cuenca-Asensi, and A. Martinez-Alvarez, "Softerror mitigation for multi-core processors based on thread replication," in *Proc. IEEE Latin Amer. Test Symp. (LATS)*, Mar. 2019, pp. 1–5.

[33] A. Waterman, Y. Lee, D. A. Patterson, and K. Asanovi, "The RISC-V instruction set manual. Volume 1: User-level ISA, version 2.0," Dept Elect. Eng. Comput. Sci., California Univ Berkeley, Berkeley, CA, USA, Tech. Rep., 2014.

[34] D. Rossi, F. Conti, A. Marongiu, A. Pullini, I. Loi, M. Gautschi, G. Tagliavini, A. Capotondi, P. Flatresse, and L. Benini, "PULP: A parallel ultra low power platform for next generation IoT applications," in *Proc. IEEE Hot Chips Symp. (HCS)*, Aug. 2015, pp. 1–39.

[35] D. Kanter, "RISC-V offers simple, modular ISA," Tech. Rep., 2016.

[36] D. A. Santos, L. M. Luza, L. Dilillo, C. A. Zeferino, and D. R. Melo, "Reliability analysis of a fault-tolerant RISC-V system-on-chip," *Microelectron. Rel.*, vol. 125, Oct. 2021, Art. no. 114346.

[37] L. Blasi, F. Vigli, A. Cheikh, A. Mastrandrea, F. Menichelli, and M. Olivieri, "A RISC-V fault-tolerant microcontroller core architecture based on a hardware thread full/partial protection and a thread-controlled watch-dog timer," in *Proc. Int. Conf. Appl. Electron. Pervading Ind., Environ. Soc.* Cham, Switzerland: Springer, 2019, pp. 505–511.

[38] I. Silva, O. D. E. Santo, D. D. Nascimento, and S. X.-D. Souza, "Cevero: A soft-error hardened SoC for aerospace applications," in *Anais Estendidos do X Simposio Brasileiro de Engenharia de Sistemas Computacionais*. SBC, 2020, pp. 121–126.

[39] S. Gupta, N. Gala, G. S. Madhusudan, and V. Kamakoti, "SHAKTI-F: A fault tolerant microprocessor architecture," in *Proc. IEEE 24th Asian Test Symp. (ATS)*, Nov. 2015, pp. 163–168.

[40] W. F. Heida, "Towards a fault tolerant RISC-V softcore," Ph.D. thesis, Delft Univ. Technol. Delft, The Netherlands, 2016. [Online]. Available: http://resolver.tudelft.nl/uuid: cee5e97b-d023-4e27-8cb6-75522528e62d

[41] A. Ramos, A. Ullah, P. Reviriego, and J. A. Maestro, "Efficient protection of the register file in soft-processors implemented on Xilinx FPGAs," *IEEE Trans. Comput.*, vol. 67, no. 2, pp. 299–304, Feb. 2018.

[42] A. E. Wilson and M. Wirthlin, "Neutron radiation testing of fault tolerant RISC-V soft processor on Xilinx SRAM-based FPGAs," in *Proc. IEEE Space Comput. Conf. (SCC)*, Jul. 2019, pp. 25–32.

[43] L. A. Aranda, N.-J. Wessman, L. Santos, A. Sánchez-Macián, J. Andersson, R. Weigand, and J. A. Maestro, "Analysis of the critical bits of a RISC-V processor implemented in an SRAM-based FPGA for space applications," *Electronics*, vol. 9, no. 1, p. 175, Jan. 2020.

[44] A. Ramos, R. G. Toral, P. Reviriego, and J. A. Maestro, "An ALU protection methodology for soft processors on SRAM-based FPGAs," *IEEE Trans. Comput.*, vol. 68, no. 9, pp. 1404–1410, Mar. 2019.

[45] A. E. Wilson and M. Wirthlin, "Fault injection of TMR open source RISC-V processors using dynamic partial reconfiguration on SRAM-based FPGAs," in *Proc. IEEE Space Comput. Conf. (SCC)*, Aug. 2021, pp. 1–8.

[46] A. Cheikh, S. Sordillo, A. Mastrandrea, F. Menichelli, and M. Olivieri, "Efficient mathematic accelerator design coupled with an IMT RISC-V microprocessor," in *Applepies* (Lecture Notes in Electrical Engineering), vol. 627. Cham, Switzerland: Springer, 2020, pp. 505–511.

[47] N. George, C. R. Elks, B. W. Johnson, and J. Lach, "Transient fault models and AVF estimation revisited," in *Proc. IEEE/IFIP Int. Conf. Dependable Syst. Netw. (DSN)*, Jun. 2010, pp. 477–486.

[48] G. Abich, J. Gava, R. Garibotti, R. Reis, and L. Ost, "Applying lightweight soft error mitigation techniques to embedded mixed precision deep neural networks," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 68, no. 11, pp. 4772–4782, Nov. 2021.

[49] G. A. Reis, J. Chang, N. Vachharajani, R. Rangan, D. I. August, and S. S. Mukherjee, "Design and evaluation of hybrid fault-detection systems," in *Proc. 32nd Int. Symp. Comput. Archit. (ISCA)*, 2005, pp. 148–159.

[50] M. Barbirotta, A. Mastrandrea, F. Menichelli, F. Vigli, L. Blasi, A. Cheikh, S. Sordillo, F. Di Gennaro, and M. Olivieri, "Fault resilience analysis of a RISC-V microprocessor design through a dedicated UVM environment," in *Proc. IEEE Int. Symp. Defect Fault Tolerance VLSI Nanotechnol. Syst. (DFT)*, Oct. 2020, pp. 1–6.

[51] Xilinx Corporation. *Soft Error Mitigation (SEM) IP Cores Perform SEU Detection, Correction, and Classification for Configuration Memory*. [Online]. Available: https://www.xilinx.com/products/intellectual-property/sem.html

MARCELLO BARBIROTTA received the master's (Laurea) degree (cum laude) in electronics engineering from the Sapienza University of Rome, Italy, in 2020, where he is currently pursuing the Ph.D. degree with the Department of Information Engineering, Electronics and Telecommunications. His current research interests include analysis techniques and models for fault resilience within digital microprocessor architectures, targeting RISC-V cores, and microcontrollers.

ABDALLAH CHEIKH received the B.S. and M.S. degrees in electrical engineering from Rafik Hariri University, Lebanon, in 2014 and 2016, respectively, and the Ph.D. degree in computer architecture (computer organization and design drove) from the Sapienza University of Rome, Italy, in 2020. He is currently a Postdoctoral Researcher at the Sapienza University of Rome. His research interests include designing, implementing, and verifying a wide range of microprocessor architectures, vector accelerators, and morphing processors.

ANTONIO MASTRANDREA received the M.S. (Laurea) degree (cum laude) in electronics engineering and the Ph.D. degree from the Sapienza University of Rome, Italy, in 2010 and 2014, respectively. He is currently a Research Assistant with the Department of Information Engineering, Electronics, and Telecommunications, Sapienza University of Rome. His current research interests include digital system-on-chip architectures and nano-CMOS circuits oriented toward high-speed computation.

FRANCESCO MENICHELLI received the M.S. (cum laude) and Ph.D. degrees in electronic engineering from the Sapienza University of Rome, Italy, in 2001 and 2005, respectively. He is currently an Assistant Professor at the Sapienza University of Rome. He has coauthored more than 40 publications in international journals and conference proceedings. His research interests include low-power digital design and, in particular, system-level and architectural-level techniques for low-power consumption, power modeling, and simulation of digital system platforms.

MAURO OLIVIERI (Senior Member, IEEE) received the M.S. (Laurea) (cum laude) and Ph.D. degrees in electronics and computer engineering from the University of Genoa, Italy. From 1995 to 1998, he was an Assistant Professor with the University of Genoa. In 1998, he joined the Sapienza University of Rome, Italy, as an Associate Professor. He is currently a Visiting Researcher at the Barcelona Supercomputing Center in the European Processor Initiative Project. He has authored more than 110 papers and a textbook in three volumes on digital VLSI design. His research interests include microprocessor core designs and digital nanoscale circuits. He has been a member of the TPC of IEEE DATE. He was the General Co-Chair of IEEE/ACM ISLPED'15. He is an Evaluator of the European Commission in the ECSEL Joint Undertaking.

Open Access funding provided by 'Università degli Studi di Roma "La Sapienza"' within the CRUI CARE Agreement