electronics

Article

Fault-Tolerant Hardware Acceleration for High-Performance Edge-Computing Nodes

Marcello Barbirotta *,†, Abdallah Cheikh, Antonio Mastrandrea, Francesco Menichelli, Marco Angioli, Saeid Jamili and Mauro Olivieri *,†

Department of Information Engineering, Electronics and Telecommunications (DIET), "La Sapienza" University of Rome, Via Eudossiana, 18, 00184 Rome, Italy; abdallah.cheikh@uniroma1.it (A.C.); antonio.mastrandrea@uniroma1.it (A.M.); francesco.menichelli@uniroma1.it (F.M.); marco.angioli@uniroma1.it (M.A.); saeid.jamili@uniroma1.it (S.J.)

* Correspondence: marcello.barbirotta@uniroma1.it (M.B.); mauro.olivieri@uniroma1.it (M.O.); Tel.: +39-06-4458-5557 (M.B. & M.O.)

† These authors contributed equally to this work.

Abstract: High-performance embedded systems with powerful processors, specialized hardware accelerators, and advanced software techniques are all key technologies driving the growth of the IoT. By combining hardware and software techniques, it is possible to increase the overall reliability and safety of these systems by designing embedded architectures that can continue to function correctly in the event of a failure or malfunction. In this work, we fully investigate the integration of a configurable hardware vector acceleration unit in the fault-tolerant RISC-V Klessydra-fT03 soft core, introducing two different redundant vector co-processors coupled with the Interleaved-Multi-Threading paradigm on which the microprocessor is based. We then illustrate the pros and cons of both approaches, comparing their impacts on performance and hardware utilization with their vulnerability, presenting a quantitative large-fault-injection simulation analysis on typical vector computing benchmarks, and comparing and classifying the obtained results. The results demonstrate, under specific conditions, that it is possible to add a hardware co-processor to a fault-tolerant microprocessor, improving performance without degrading safety and reliability.

Keywords: fault-tolerant hardware accelerators; hardware accelerators; fault injection; microprocessors; RISC-V

check for updates

Citation: Barbirotta, M.; Cheikh, A.; Mastrandrea, A.; Menichelli, F.; Angioli, M.; Jamili, S.; Olivieri, M. Fault-Tolerant Hardware Acceleration for High-Performance Edge-Computing Nodes. *Electronics* **2023**, *12*, 3574. https://doi.org/10.3390/electronics12173574

Academic Editor: Shinichi Yamagiwa

Received: 15 July 2023
Revised: 18 August 2023
Accepted: 22 August 2023
Published: 24 August 2023

1. Introduction

High-performance embedded systems are becoming increasingly prevalent in daily life, because they are used in various applications, from smartphones and laptops to automotive and aerospace industries, industrial automation, medical equipment, and telecommunications. Since power consumption is a critical aspect of the domain of embedded systems conceived to be powered by batteries, they must be designed to consume minimal power to reduce cost and improve portability. For these reasons, one of the key challenges in embedded systems is balancing the need for high performance with low power consumption. This challenge is addressed using hardware acceleration units, which can process a large number of data in real time and offload computationally intensive and specific tasks from the main processor to specialized hardware, allowing for significant power savings while maintaining a high level of performance.

In recent years, hardware acceleration has become increasingly popular in the Internet of Things (IoT), Artificial Intelligence (AI), and powerful embedded systems [1]. Hardware accelerators implemented as digital signal processors (DSPs) or developed inside Field-Programmable Gate Arrays (FPGAs) can accelerate a wide range of applications [2]. They can be used to implement security features critical in the IoT and embedded systems; they can perform complex encryption and decryption operations in real time that are essential for communication [2]; they can be used to perform complex image processing tasks, such as object detection, face recognition, and image compression, which would otherwise be impossible with a traditional processor [3]. Finally, they can also perform complex matrix multiplications and convolution operations in real time, which are essential for deep learning algorithms, with many benefits among all the high-performance embedded systems domains. Since the IoT world has transitioned from cloud computing to edge computing, it is populated with systems capable of quickly pushing computation and data storage closer to the source of data rather than relying on a centralized data center to cope with the limited network bandwidth and to minimize the high computational demand required to compute the large number of data acquired by edge nodes [4]. For this reason, the edge-computing world is starting to be populated by powerful embedded system devices capable of fulfilling all these requirements.

On the fabrication technology side, the realization of ever smaller and more performing devices has led the research world to various problems linked to CMOS scaling and safety [5]. The minimum feature size and voltage-margin reduction increase safety-critical faults in digital chips, along with augmented statistical process variations [6]. Ionizing particles and high-energy radiation affect normal system functions, creating faults during its operation, making it difficult for full protection and high performance to coexist. Owing to their extensive use and criticality, high-performance embedded systems must be designed to be highly reliable and fault-tolerant, allowing for the correct functioning of the system even in the event of a fault, especially if they operate in harsh environments for mission-critical applications such as military, space, and avionic applications or are used for safety-critical applications such as automotive or medical applications, everything achieved using hardware- or software-redundant systems, error detection and correction, and recovery mechanisms.

In this study, we extend the preliminary work introduced in [4] by merging high-performance edge computing and fault tolerance (FT), demonstrating and quantitatively analyzing how to achieve high-performance vector acceleration on a fault-tolerant microprocessor core in all applications where both safety and high performance are required.

We aim to start from the use of an Interleaved-Multi-Threading (IMT) core modified to achieve fault-tolerant execution [7–9] thanks to the implementation of a new FT technique called Buffered Triple-Modular Redundancy (TMR), able to integrate both TMR and temporal redundancy, adding the support of a vector acceleration unit [10,11] and describing all the performance in terms of fault tolerance and reliability to obtain a completely fault-tolerant accelerated core.

The contributions of the proposed study are as follows:

- Filling the gap in the literature about fault-tolerant hardware acceleration in edge-computing devices.
- Demonstrating how a full-TMR hardware accelerator works, describing its architecture, and proving its functionalities with extensive fault-injection (FI) tests.
- Demonstrating that thanks to the inherent behavior of an Interleaved-Multi-Threading structure, it is possible to convert a full-TMR hardware accelerator into a single-buffered accelerator without degrading system reliability, decreasing hardware overhead, cost, and power consumption.

The rest of the paper is organized as follows: Section 2 discusses the related works on FT in the IoT and edge-computing systems; Section 3 outlines the proposed architecture ideas and methodology; Section 4 presents FPGA-synthesis comparisons; Section 5 presents the FI tests and the results obtained in terms of fault resilience; Section 6 reports the comparison with existing FT techniques on RISC-V processors. Finally, Section 7 summarizes the conclusions of our study.

2. Related Works

In the literature, a considerable number of fault-tolerant applications have been proposed for the IoT, and edge and cloud computing. However, these do not provide the presence "on the edge" of redundant hardware accelerators. The authors in [12] present an interesting review about FT in IoT systems, classifying many FT-IoT works according to the techniques used to obtain FT. Particularly important is the one based on Cloud, where the impact of a failed component is minimized within an integrated fog–cloud platform, with Microservices being able to implement small applications with single responsibility giving support to FT for real-time fault detection, or layered views, where systems are viewed as a complex heterogeneous entity that can be decomposed into interacting layers ranging from 3 to 5 and that communicates via software. Microservices are also explained in [13]. They are used to build a specific framework with two fault-tolerant features, real-time FT detection and online machine learning, to detect fault patterns and mitigate faults before they are activated, and they have a scalable architecture with individual responsibilities for each micro-application. The work in [14] presents a way to perform energy-efficient and low-overhead fault mitigation in the IoT, using ECC for single and multiple faults, with SEC DED TAEC 6AED techniques to add redundancy in the IoT and Network on Chip (NoC), evaluating all these techniques with respect to joint-crosstalk-avoidance multiple-error correction (JCAMEC) and joint-crosstalk multiple-error correction (JMEC). The authors in [15] present a military view of the IoT (MIoT) for mission-critical scenarios, in which the same techniques that can be applied in safety-critical systems are used, providing virtualization and replication capabilities with cloud or fog platforms able to reduce latency and increase the resilience of all MIoT layers, integrating advanced machine learning techniques for fault detection and diagnosis.

Other mechanisms are based on neural networks or customized for neural network applications. The work [16] presents the Selective TMR tool, which analyzes the sensitivity in a neural network architecture with respect to the overall accuracy and triples the most sensitive parts, increasing functional safety without resulting in full TMR. The work [17] presents a prediction of faults to obtain preemptive migration decisions in case of a future failure of the system, while the authors in [18] present the OR-ML approach to enhance the reliability of "special purpose" ML accelerators by opportunistically exploring the chances of runtime redundancy provided by neighboring PEs without adding computing resources, significantly reducing implementation overhead. Further, in the field of special-purpose accelerators, the authors in [19] present fault-aware pruning (FAP) and retraining (FAP + T). They declare that it is possible to modify the deep neural network architecture to adapt it to the faults without deleting them by re-executing the test, spending extra time overhead.

Over the years, it was taken for granted that node redundancy and network contingencies were the best solutions [20], and a lot of work has been performed on high-level structures, software or hybrid cloud-computing frameworks [21], and virtualization layers provided by containers with checkpointing mechanism in case of a node failure. However, nowadays, with the increasing number of edge devices, this solution becomes not feasible on edge-computing nodes [22].

The focus of the proposed study is the definition of an FT microarchitecture for those edge-computing applications requiring both safety and high performance. In this work, we deal with the edge node, trying to obtain redundancy and protection directly inside the node and lightening the load on subsequent higher layers of the system, which are also necessary. We, therefore, deal with the hardware aspects of the single node, equipped with a general-purpose vector co-processor capable of performing any vector arithmetic operation in an FT way, differently from most of the FT vector accelerators created for specific special-purpose applications.

Different techniques have been proposed for FT in IoT systems [12]: time redundancy at the instruction and task levels, with the replication of tasks to obtain redundancy; active or passive replication with multiple processors in fog or cloud, which executes the same process concurrently or executes tasks only in case of fault, respectively; network control, where a cluster head makes requests to the other nodes and confirms a failure in case of no reply message; distributed recovery block, with lock-step execution between nodes and the testing of the results, which is performed only if the test passed. To the best of our knowledge, little work has been performed to protect a single edge node, especially if it has hardware acceleration features.

This work builds on the preliminary idea presented in [4]. With respect to [4], the work presents the following:

- An in-depth analysis of the idea of the replicated VCU, with an expanded and detailed description;
- The introduction of the hybrid-VCU mode, assuming and analyzing the impact of memory-scrubbing techniques within the accelerator scratchpad memory;
- The novel detailed analysis of fault-resilience performance on algorithms of practical interest in hardware acceleration, such as FFT and Matmul.

3. Proposed Approach

3.1. Methodology

The framework for this study is the Klessydra-fT13 architecture, a fault-tolerant 32-bit RISC-V IMT soft processor integrated inside the PULPino [23] open-source System-on-Chip architecture. The processor is composed of a fault-tolerant non-accelerated scalar core, resembling Klessydra fT03 [7–9], tightly coupled with a fault-tolerant configurable accelerating co-processor unit (Figure 1).

The processor was implemented as an RTL design and synthesized on the Vivado platform. Our analysis focused on the effect of SEU faults on sequential cells in the hardware. The RTL files were integrated in an ad hoc Universal Verification Methodology (UVM) environment, as it has dedicated support for SEU fault injection in the Flip-Flop (FF) cells of the design. In order to be able to inject faults in the RTL code corresponding to FFs, an automatic pre-processing routine was run at the RTL to identify all the synchronous assignments, which generate FFs in hardware implementation. The target of the present work was the validation of the vector accelerator subsystem's resilience, while the scalar core was already validated in previous works, so we concentrated on the fault-injection campaign on the accelerator microarchitecture.

The FI campaign was defined according to the Time Frame simulation methodology (described in Section 5.1), which allows for the calculation of an upper bound to the application program's failure probability assuming that an SEU fault hits the processor. The analysis was performed on individual units of the accelerator microarchitecture, so as to obtain a detailed identification of the critical parts of the design for FT. The analysis was further enhanced by integrating memory scrubbing in the simulation to evaluate the impact of this feature on the overall FT. Further details of the research methodology steps are described in the corresponding sections.

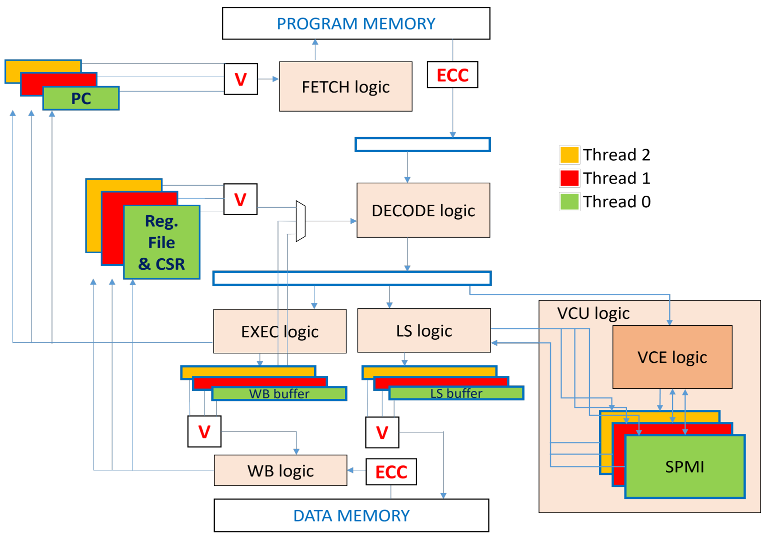

Figure 1. Klessydra-fT13 microarchitecture.

3.2. Fault-Tolerant Scalar-Core Microarchitecture

IMT cores are not immediately usable for fault-tolerant applications, because they do not rely on replicated processor cores, such as multi-core architectures or multiple Functional Units (FUs), or they do not rely on Simultaneous-Multi-Threading (SMT) architectures, and the results are not available at the same time [4]. We exploited the IMT architecture to merge spatial redundancy and temporal redundancy, developing the Buffered TMR technique, in which three threads are instances of the same program maintaining their state in redundant registers (spatial redundancy) with shared combinational logic and executing interleaved instructions with one-cycle time distance between any two instructions belonging to different threads (temporal redundancy). We modified a non-hardened core [10,11], proving the intrinsic FT capability of an IMT architecture with three identical threads, each having its own register file (RF), program counter (PC), and control/status registers (CSRs). The Buffered TMR paradigm implies using specific buffer registers that can maintain the state produced by the corresponding thread, waiting for the latest execution of the three threads (Thread 0) before voting on the results. In the pipeline on the left side of Figure 1, it is possible to highlight the use of buffer registers within the units that produce data, such as the Execution Unit (EXE) or Load–Store Unit (LSU). Write-back voting can occur when Thread 0 reaches the write-back phase as well as load/store operations, which are performed by Thread 0 in a voting manner. Because of the buffer registers in the LSU, it is possible to prevent replicated load/store access to the same location, consume less power, and avoid inconvenient behavior when reading peripherals. The interested reader may refer to [7,8] for additional details and performance evaluation of fault-tolerant scalar cores.

3.3. Fault-Tolerant Vector Co-Processor

The primary targets of this work are the application of the Buffered TMR concept to the vector acceleration unit and its evaluation. The Vector Co-Processing Unit (VCU) in Klessydra processors is a highly configurable general-purpose vector accelerator [10,11] capable of processing up to 256 bit of data in a single cycle. It comprises two main components: the Vector Co-Processing Engine (VCE), which executes vector instructions, and the Scratchpad Memory Subsystem (SPMI), which locally stores intermediate results of the computation kernel, similarly to a vector register file. The VCU can be customized using a variety of parameters, such as the number of SPMs per SPMI, SPM size, Single-Instruction Multiple-Data (SIMD) width, and execution paradigm, which usually enables Multiple-Instructions Multiple-Data (MIMD) capabilities by dedicating a co-processor to each thread of execution in the application. In this study, we particularly focus on the execution paradigm parameter, as customizing its usage can affect the FT features of the co-processor. There are three main execution paradigm configurations in the vector co-processor: a single VCU shared by all threads, a distinct VCU for each hart, and a hybrid VCU. The first configuration offers no redundancy and thus is not suitable for FT purposes. In the scope of fault-tolerant accelerators, we are particularly interested in the latter two schemes, because they maximize instruction-level parallelism (ILP) and offer some degree of potential redundancy in the execution. For details on the internal operation of the vector accelerators used in this work, the interested reader may refer to [10,11].

In the replicated-VCU scheme, each thread in the fT13 core has its own VCE and SPMI. When a vector instruction arrives at the decoding stage, it is directed to the VCU corresponding to its hardware-thread identifier (hartID). The instruction then requests the data from its own SPMI, processes the instruction, and writes it back to its own destination SPM. A stall in a thread can only occur when its own VCU is busy. Figure 2 illustrates this scheme. In a non-hardened accelerator, the scheme allows for the parallel execution of vector instructions, whereas in fT13, it is exploited to obtain redundant execution of the same vector instruction among three parallel accelerators. Therefore, the first characteristic of this structure is the concept of spatial redundancy at the architectural level. The thread-dedicated VCUs can implement a hardware replication typical of multi-core fault-tolerant systems, offering instruction execution redundancy at the cost of reducing the MIMD capabilities of the vector co-processor, which can only work in SIMD mode. Since the three SPM banks work in parallel and independently from each other, yet each of them writes the same data owing to the temporal redundancy of the instructions, it would be theoretically possible to further increase the architectural redundancy using voting when the three threads read data from all of the three scratchpads simultaneously. However, since the SPM bank works with a single read port, this solution is not feasible for hardware utilization nor is the cost of implementing a writing voting mechanism. We opted for a solution in which each thread writes its results on its own SPM only, and any SPM writing errors are corrected when writing back the results in the data memory using redundancy in the LSU [8]. While this mechanism can solve isolated inconsistencies in the final data, it does not prevent error accumulation and propagation within the SPM banks. As a consequence, errors in an SPM related to one thread could add up to other errors in another SPM related to another thread, thereby preventing majority voting. Depending on the statistical frequency of such situations, ECC protection or a memory-scrubbing mechanism may be necessary, capable of autonomously checking and correcting all the SPM banks, as it already happens within the data memory and instruction memory of the scalar core.

In the hybrid-VCU scheme, each hardware thread in the fT13 core has its own SPMI, but all share Functional Units in the VCE. This scheme allows for the parallel execution of vector instructions as long as the different threads use different Functional Units; otherwise, they are buffered in a queue. When a vector instruction arrives at the decoding stage, it is directed to the VCU. The instruction then requests and stores local data from/to its thread-related SPMI.

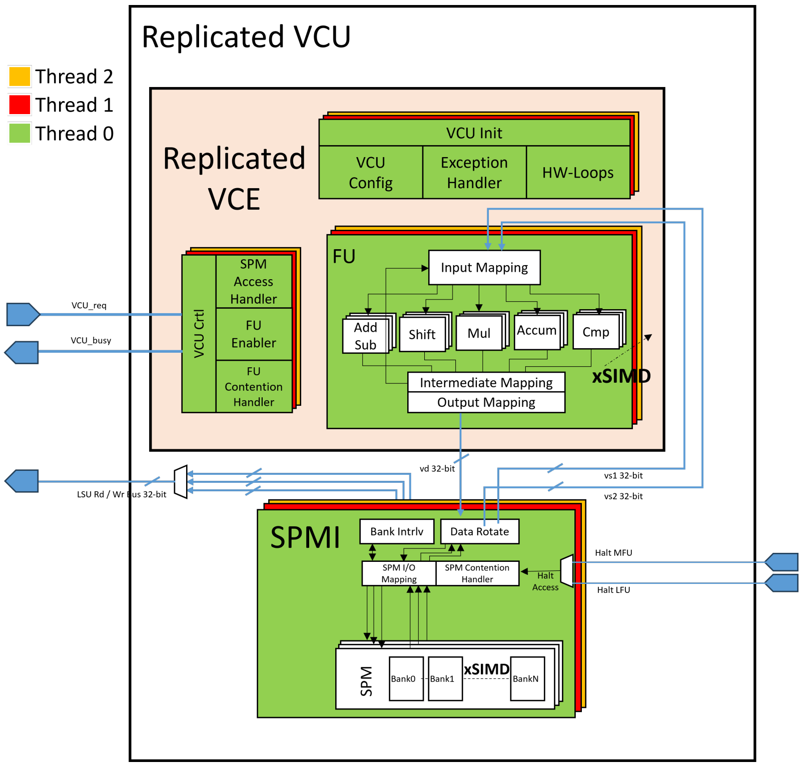

Figure 2. Replicated-VCU scheme with dedicated hardware. Once the instruction is decoded for each thread, it goes to the VCU and requests data from its own SPMI, processing the instruction inside the FU and writing back the result inside the SPM.

During vector-instruction execution, if a vector instruction of another thread arrives at the VCU, it checks if the Functional Units to be used are free, and in such a case, the thread goes ahead in parallel with the first thread, on the same accelerator yet in a different Functional Unit. Figure 3 illustrates this scheme. A key characteristic of this scheme is that harts employing redundancy in the execution of instructions always make the case where there will be contention on a busy Functional Unit in the VCU. Hence, the scheme automatically deploys temporally redundant execution of instructions, as an identical instruction is dispatched after the first instruction is executed. Since, in the hybrid-VCU scheme, each thread has its SPMI and the VCU operation is no longer symmetrically parallel, it is not possible to vote on writing back the results from the SPMs to the main memory. In this context, memory-scrubbing mechanisms are more necessary than in the previous cases and should be implemented after each memory write operation.

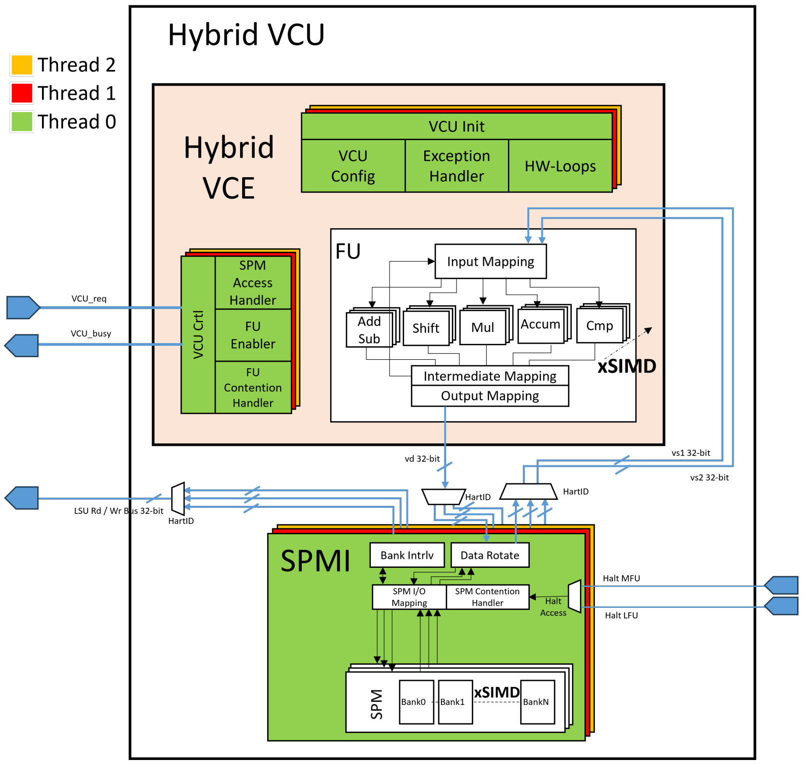

Figure 3. Hybrid-VCU scheme with shared hardware. Once the instruction is decoded for each thread, it goes to the VCU and requests data from its own SPMI, processing the instruction inside the shared free FU, waiting in case of busyness, and then writing back the result inside the SPM.

4. Impact on Hardware Resources

Table 1 shows that the hardware configuration of Klessydra-fT13 in both approaches, compared with the one presented for the non-hardened Klessydra-T13, results in handling only SIMD mode, caused by the single-thread behavior of Klessydra-fT13, which loses the MIMD capabilities due to instruction replication among the threads. From the hardware point of view, synthesis results from Vivado 2022 targeting a Genesys2 board show larger hardware utilization compared with the Klessydra-T13 version, as well as a decrease in hardware acceleration performance and frequency due to redundancy and voting blocks. By comparing the hardware utilization of the two accelerators, it is easy to observe that hardware resources can be saved at the expense of the FT level by choosing the hybrid-VCU scheme, with respect to the replicated VCU.

Table 1. Synthesis@Vivado 2022 on Genesys2 board and performance results.

Architecture		**FPGA Synthesis Results**					**Perf**	
Core	HW	FF	LUT	B-RAM	DSP	LUT-RAM	Freq. (Mhz)	DLP
T13	MIMD + SIMD	4712	15,943	18	19	264	131.7	2
		6753	25,089	18	31	264	120	4
		10,854	43,419	36	55	264	105.1	8
fT13 repl.	SIMD	9017	20,174	48	31	0	108	2
		11,671	33,250	48	55	0	102	4
		17,006	53,198	48	103	0	90	8
fT13 hyb.	SIMD	7795	20,658	48	15	0	105	2
		9177	27,230	48	23	0	97	4
		12,117	50,913	48	39	0	88	8
T03	-	1418	4281	0	7	176	221.1	-
fT03	-	4910	6670	0	0	0	200	-

5. Validation

To validate the resilience of both architectures, we implemented a Time Frame Spanning fault-injection (FI) simulation [24], within a Universal Verification Methodology (UVM) environment able to simulate SEU faults hitting synchronous registers in the design. We, furthermore, implemented a classical random Monte Carlo FI simulation specifically devoted to analyzing the potential improvement given by an ideal memory-scrubbing unit in the SPMs of the VCUs.

In the analysis, we used two common machine learning (ML) and digital signal processing (DSP) application algorithms as the benchmark workload, namely, FFT and Matrix multiplication. Each FI experiment covers the entire program execution, injecting logic value flips in target bits previously selected within the processor design as an Architecturally Correct Execution (ACE) bit [25]. By definition, an ACE bit is a bit of the architecture in which a fault in a specific clock cycle may cause a program failure.

5.1. Failure Probability Estimation with Time Frame Spanning FI

The Time Frame Spanning methodology [24] divides the whole execution time into M different intervals called time frames. Unlike classical Monte Carlo FI methods, the approach performs the deterministic injection of many faults within the architecture in a specific time frame within the total execution time for each target bit in the hardware architecture. An ad hoc simulation is launched for each target bit, injecting at least one fault every 40 clock cycles for each of the M time frames, resulting in an extremely high-fault-rate analysis. Table 2 shows the Time Frame Spanning setup planned to analyze the core under FI. We decided to only target the internal bits of the VCU, as the other units were previously studied in different works [7,8], comparing the results with those obtained on the non-hardened Klessydra-T13 processor, containing the same vector accelerator.

Table 2. Test setup with clock cycles required to complete the benchmarks; number of frames used by the Time Frame Spanning techniques [24] and the total faults per frame, with an average of 1 fault every 40 clock cycles.

	fft			**Matmul**	
Core	**T13**	**fT13**	**fT13-hyb**	**T13**	**fT13**
Total clock cycles	91182	95925	144639	220490	227344
# frames	10			10	
Faults/frame	225	237	357	545	561
Deterministic fault rate	1 every 40 cycles			1 every 40 cycles	

Figure 4 shows the FI results on the FFT benchmark for all the architectures under analysis. Comparing the FI results of both processors highlights how the redundant-VCU case has greater FT properties than the hybrid-VCU case. The Klessydra-fT13-hybrid processor has a much lighter structure than the fT13-replicated one, with single FUs per thread. Having many shared hardware resources, it is inherently more prone to failures. Nonetheless, owing to the Buffered TMR paradigm, we found that this shared version of the VCU may exhibit relatively good FT features. Observing the results, it can be seen that the green bars of the replicated-VCU case and the blue bars of the hybrid VCU represent the advantage obtained in terms of fault tolerance. In the replicated case, the advantage is represented by the reduction in the total failure rate, obtained as the weighted average based on the number of bits in each register, from 40% to about 3.7%. In comparison, this advantage is minor in the hybrid case, with a decrease from 31% to about 22%. The lowest level of failures related to the T13 hybrid case is due to the fault-masking effect. For the non-hardened T13 processor, it is possible to hypothesize that some faults are masked by the execution of the last thread in interleaving order. With three threads interleaved in temporal execution, faults that occur during Thread 2 or Thread 1 can be masked by the execution of Thread 0, resulting in a low failure rate. Thread 0 is used to write down the obtained result into the same scratchpad memory location of the previous threads, deleting any previous error. Forcing high fault rates avoids the masking effects that lead to observing errors inside the scratchpad memory, which in turn are written inside the data memory despite the ability to hide them. In the T13-hybrid case, to further break down the masking problem, the test was only performed on the single Thread 0, reporting failures only for its related signals. The minor advantage of fault tolerance in the hybrid case is due to the Functional Units shared by all the threads that do not allow for the parallel execution of the instructions. There is now no complete spatial redundancy, and the Buffered TMR technique guarantees the only redundancy level. For both FFT tests, comparisons are made with the replicated-VCU and hybrid-VCU versions having memory scrubbing, bringing the weighted failure rates to 1.53% and 1.18% for replicated and hybrid, respectively. As a proof of concept, we also developed an analysis for the system with memory scrubbing. For both cases, it can be noted that the impact on the use of memory scrubbing is considerable and almost comparable, pointing out how the periodic control of memory systems in FT architectures is a critical and essential task.

Figure 5 shows the breakdown of the system failure probabilities associated with different registers inside the VCU for the Matmul benchmark, for the best-performing fT13-replicated VCU architecture. In this figure, we can recognize the same characteristics described for Figure 4. The non-protected architecture experiences failure rates close to 90% in most registers. Similar to Klessydra-fT03 [7,8], the protection level is also extended with temporal redundancy with the use of the same instructions among different threads, providing a dual protection layer (spatial and temporal) on all operations that activate the co-processor [4], saving the output produced by each thread within the individual, dedicated SPMs. Figure 5 also shows that each SPM can be treated as a result buffer in the context of a Buffered TMR paradigm. Voting among the SPMs during LSU operations leads to the deletion of most of the faults that would otherwise enter the main data memory. Figure 5 points out again how SPM scrubbing leads to enormous benefits even in the replicated-VCU case compared with the non-scrubbed case.

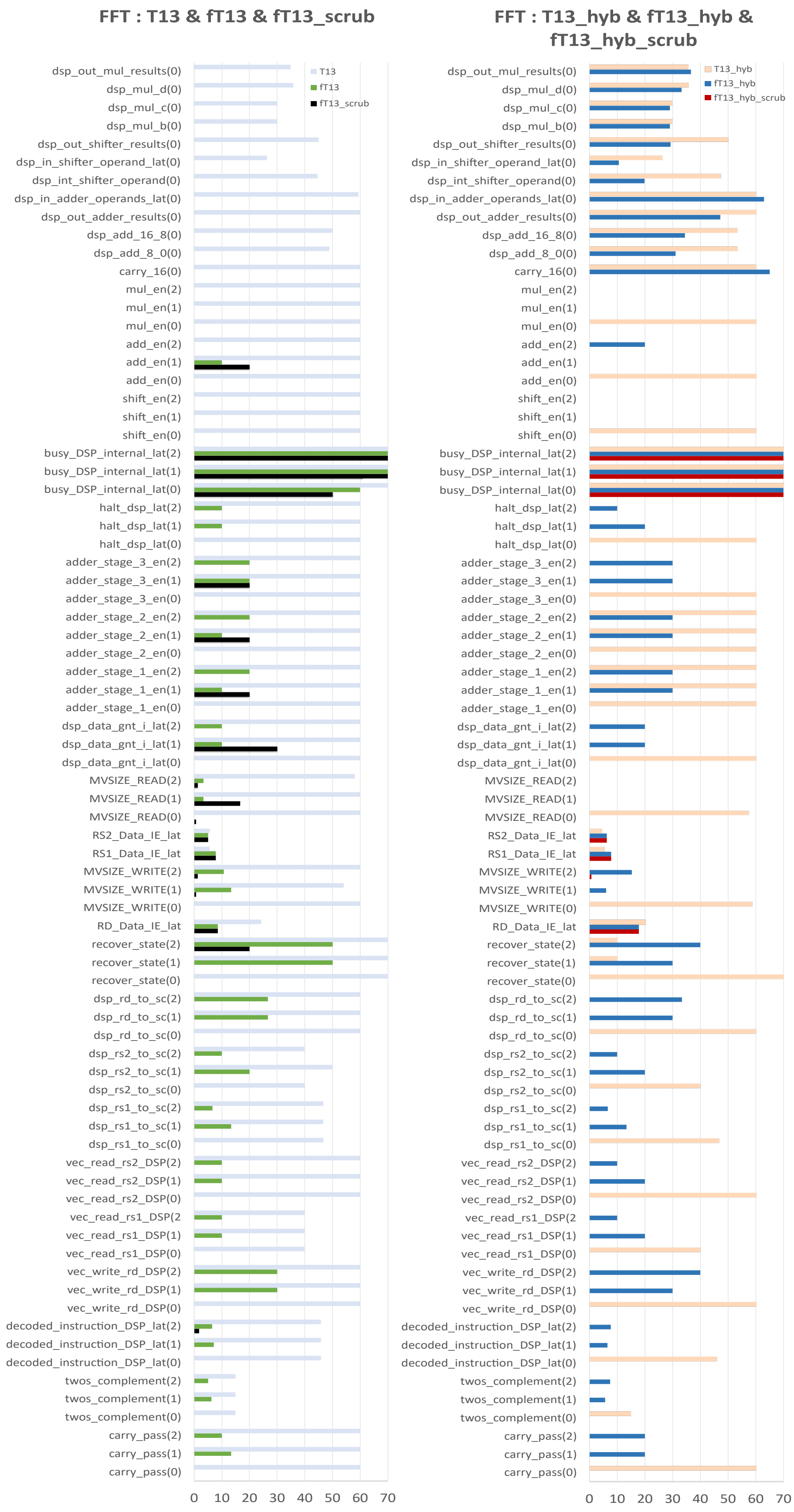

Figure 4. FFT FI simulation results for T13, fT13, and scrubbed fT13, as well as T13-hybrid, fT13-hybrid, and scrubbed fT13-hybrid, showing the FT improvements and differences between redundant and hybrid architectures.

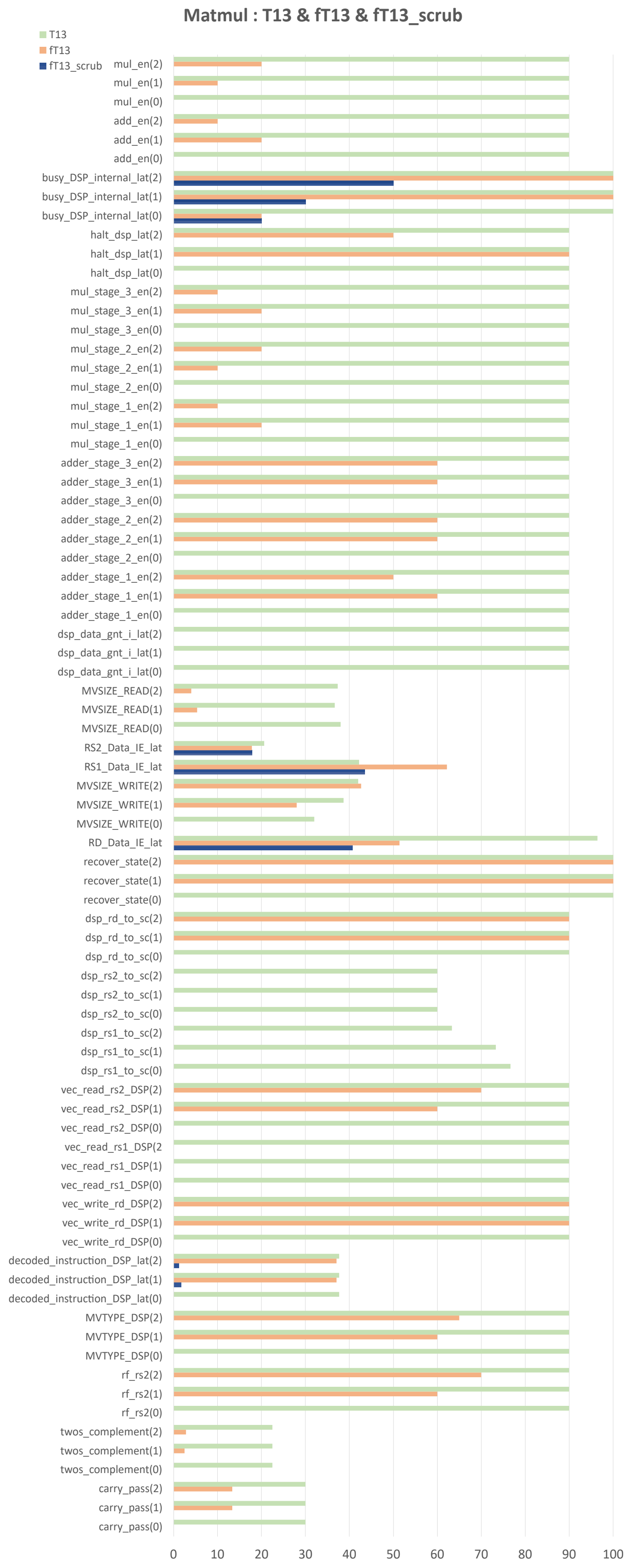

Figure 5. Matmul FI simulation results for T13, fT13, and scrubbed fT13, all with replicated-VCU architecture, showing the FT improvements due to the union of the replicated-VCU architecture and the redundant fT03 processor.

5.2. Monte Carlo FI Simulation to Analyze Memory-Scrubbing Impact

We performed an additional, conventional, random Monte Carlo FI simulation to better highlight the impact of SPM scrubbing. In this simulation, each bit is tested individually, and for each test run, a certain number of random faults is inserted, gradually increasing. There is no interest in seeing the failure rate of the individual bits, and the test results of all the architecture bits are averaged among them and reported in a graph. Without loss of generality, we report the results for the fT13-hybrid architecture, with the other case being substantially equivalent. Figure 6 shows the Monte Carlo FI process with randomly injected faults in a range from 10 up to 2000 for the FFT benchmark, for each of the 2154 target bits in the VCU. Comparing a non-hardened T13-hybrid processor and the fT13-hybrid processor shows a constantly lower failure rate in the latter. For a low fault rate, this solution results in good FT performance.

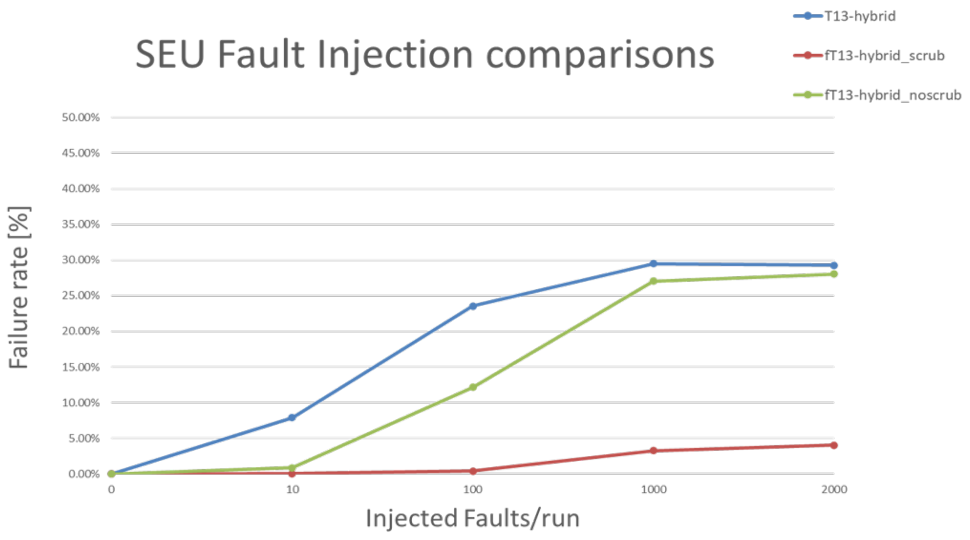

Figure 6. Random Monte Carlo simulation results inside the hybrid-VCU scheme, with faults in a range from 10 up to 2000 for the FFT benchmark on a total of 2154 target bits from the DSP unit.

Comparing Figure 6 and Figure 7, it is possible to note that most of the failures are related to memory inconsistencies with respect to the golden reference, due to incorrect values inside the scratchpad memories, which, if not corrected by a voting system, inevitably lead to incorrect values in the data memory. As proof of concept, we analyzed the impact of an ideal memory-scrubbing mechanism that checks the SPM banks and periodically corrects wrong values when scratchpads are not in use. The orange line in Figure 6 represents the fT13-hybrid processor with SPM scrubbing, which results in a lower failure rate. This shows the theoretical optimal performance that we would obtain by having the ideal hardware units capable of performing perfect SPM scrubbing (Figure 7). Practical implementations with corresponding hardware impacts will be addressed in future works.

Overall, the reliability of the system stays around 1–2% with the support of memory-scrubbing techniques, with a decrease from 17,006 to 12,117 in FFs and from 53,198 to 50,913 in LUTs for the SIMD 8 configuration, thanks to the shared Functional Units. While we do not have reliable, precise data on power consumption, we expect that the significant lower use of hardware resources also results in a decrease in power consumption.

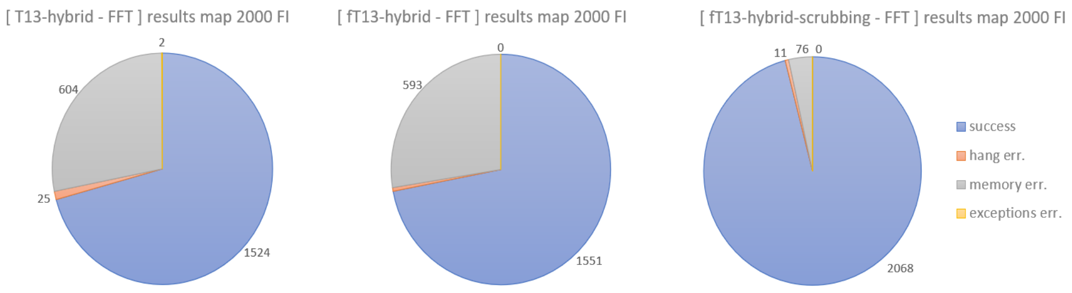

Figure 7. Random Monte Carlo error classification related to Figure 6. Memory errors decrease with the use of memory-scrubbing techniques.

6. Comparison with Existing Fault-Tolerance Techniques

Since our experiments target the RISC-V instruction set architecture (ISA) [26], RISC-V-compliant fault-tolerant cores are the reference architectures for performance comparison, so that the results may not be influenced by the use of different ISAs [8]. In order to compare different FT designs, it is possible to refer to the normalized failure percentage (NFP) figure, obtained as the ratio between the failure percentage of the protected microarchitecture and the failure percentage of the unprotected version of the same microarchitecture subjected to the same FI campaign [27]. Table 3 reports the outcome of the analysis. In [28], the authors present the BL-TMR software suite, which can analyze a Xilinx Vivado netlist and add triple redundancy to critical nodes of the RISC-V Taiga processor implemented on a Kintex Ultrascale device, resulting in the hardware resources of 3X FFs and 5.64X LUTs. Based on the declared improvement of 24X in the average time between failures, we can estimate an NFP of 1/24 = 4.1% by assuming a uniform time distribution of radiation-induced faults. In [27], the authors protected the Rocket RISC-V processor with a distributed TMR technique, exhibiting 3X overhead in FFs and LUTs and 2.3% NFP. In contrast, the work reported in [29] exploits the idea that only the most statistically frequent ALU operations require protection to reduce hardware overhead, obtaining NFP in a range from 40% to 5.63%, with FI affecting only the protected units. In [30], the authors use a Hamming code to protect the register file and the program counter, and TMR to protect the ALU and control logic. The modified microarchitecture was implemented on a Xilinx Zynq FPGA, utilizing 1.19X FFs and 1.70X LUTs and obtaining 7.7% NFP. In the non-accelerated Klessydra-fT03 design [8], the overhead was reported as 1.09X in FFs and 1.17X in LUTs, with an NFP of 2.3%, compared with the 1.53% NFP obtained in the presented work. The latter result is related to FI tests carried out on the VCU accelerator subsystem, which was the target of our analysis, so the actual NFP on the entire processor (Figure 1) can be estimated as an average between 1.53% and 2.3% depending on the VCU configuration and the consequent ratio between the areas of the scalar processing pipeline and of the VCU.

Table 3. Fault-tolerant RISC-V core summary divided by FT techniques with normalized failure percentage (NFP).

Work	Core	FT Techniques	FT Units	Reported FT Results
[28]	Taiga	Distributed TMR	Configuration memory	4.1 % NFP
[27]	FT Rocket	Distributed TMR	Entire microarchitecture	2.3% NFP
[29]	FT lowRISC	Application-tailored TMR	ALU	from 40% to 5.3%
[30]	Ad hoc core	Partial TMR and Hamming	ALU, PC, Reg.File, control logic	7.7% NFP
[8]	Klessydra-fT03	Buffered TMR	Reg.File, Write-Back Unit, PC, Load–Store Unit	2.9% NFP
This work	Klessydra-fT13	Buffered TMR + scrubbing	Reg.File, Write-Back Unit, PC, Load–Store Unit + VCU	1.53% NFP

7. Conclusions

This work investigated the potentials and the issues of designing fault-tolerant hardware acceleration for edge-computing devices. The first research goal mentioned in the introduction was attained by performing a comprehensive study on hardware acceleration in edge-computing devices and demonstrating how hardware techniques like Buffered TMR can improve FT using redundancy in hardware co-processors within an IMT RISC-V core for high-performance embedded systems, filling a gap in the existing literature. We described two co-processor architectures, replicated VCU and hybrid VCU, allowing us to obtain a fault-tolerant co-processor companion to a fault-tolerant scalar core. The second research goal was attained by proving all the full-TMR replicated-VCU hardware accelerator functionalities, with extensive fault-injection tests on DSP benchmarks, commenting and reporting all the results regarding hardware overhead and fault tolerance. The third research goal was reached by describing a different hybrid architecture, demonstrating that thanks to the inherent behavior of an Interleaved-Multi-Threading structure, it is possible to replace a full-TMR hardware accelerator like the replicated VCU with a single-buffered accelerator like the hybrid VCU with limited degradation of system reliability and significantly less hardware overhead, also describing the potential impact of SPM scrubbing on reducing the failure probability.

Author Contributions: Conceptualization, M.B.; Methodology, F.M.; Software, M.A.; Validation, M.B.; Formal analysis, S.J.; Investigation, A.C.; Resources, A.M.; Writing—original draft, M.B. and A.C.; Writing—review & editing, M.O.; Visualization, F.M.; Supervision, M.O.; Project administration, M.O.; Funding acquisition, M.O. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: https://github.com/klessydra.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Cardarilli, G.C.; Nunzio, L.D.; Fazzolari, R.; Panella, M.; Re, M.; Rosato, A.; Span, S. A Parallel Hardware Implementation for 2-D Hierarchical Clustering Based on Fuzzy Logic. *IEEE Trans. Circuits Syst. II Express Briefs* **2021**, *68*, 1428–1432. [CrossRef]
2. Cardarilli, G.C.; Di Nunzio, L.; Fazzolari, R.; Giardino, D.; Re, M.; Ricci, A.; Spanò, S. An FPGA-based multi-agent Reinforcement Learning timing synchronizer. *Comput. Electr. Eng.* **2022**, *99*, 107749. [CrossRef]
3. Cardarilli, G.C.; Di Nunzio, L.; Fazzolari, R.; Giardino, D.; Nannarelli, A.; Re, M.; Spanò, S. A pseudo-softmax function for hardware-based high speed image classification. *Sci. Rep.* **2021**, *11*, 15307. [CrossRef] [PubMed]
4. Barbirotta, M.; Cheikh, A.; Mastrandrea, A.; Menichelli, F.; Olivieri, M. Analysis of a Fault Tolerant Edge-Computing Microarchitecture Exploiting Vector Acceleration. In Proceedings of the 2022 17th Conference on Ph.D Research in Microelectronics and Electronics (PRIME), Villasimius, Italy, 12–15 June 2022; pp. 237–240. [CrossRef]
5. Barbirotta, M.; Mastrandrea, A.; Cheikh, A.; Menichelli, F.; Olivieri, M. Improving SET Fault Resilience by Exploiting Buffered DMR Microarchitecture. In Proceedings of the SIE 2022: 53rd Annual Meeting of the Italian Electronics Society, Pizzo, Italy, 7–9 September 2022; Springer: Berlin/Heidelberg, Germany, 2023; pp. 233–238.
6. Khalid, U.; Mastrandrea, A.; Olivieri, M. Novel approaches to quantify failure probability due to process variations in nano-scale CMOS logic. In Proceedings of the 2014 29th International Conference on Microelectronics Proceedings-MIEL 2014, Belgrade, Serbia, 12–14 May 2014; pp. 371–374. [CrossRef]
7. Barbirotta, M.; Cheikh, A.; Mastrandrea, A.; Menichelli, F.; Vigli, F.; Olivieri, M. A Fault Tolerant soft-core obtained from an Interleaved-Multi- Threading RISC- V microprocessor design. In Proceedings of the 2021 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), Athens, Greece, 6–8 October 2021; pp. 1–4. [CrossRef]
8. Barbirotta, M.; Cheikh, A.; Mastrandrea, A.; Menichelli, F.; Olivieri, M. Design and Evaluation of Buffered Triple Modular Redundancy in Interleaved-Multi-Threading Processors. *IEEE Access* **2022**, *10*, 126074–126088. [CrossRef]
9. Barbirotta, M.; Cheikh, A.; Mastrandrea, A.; Menichelli, F.; Ottavi, M.; Olivieri, M. Evaluation of Dynamic Triple Modular Redundancy in an Interleaved-Multi-Threading RISC-V Core. *J. Low Power Electron. Appl.* **2022**, *13*, 2. [CrossRef]
10. Cheikh, A.; Sordillo, S.; Mastrandrea, A.; Menichelli, F.; Olivieri, M. Efficient mathematical accelerator design coupled with an interleaved multi-threading RISC-V microprocessor. In Proceedings of the Applications in Electronics Pervading Industry, Environment and Society: APPLEPIES 2019, Pisa, Italy, 11–13 September 2019; Springer: Cham, Switzerland, 2020; pp. 529–539. [CrossRef]
11. Cheikh, A.; Sordillo, S.; Mastrandrea, A.; Menichelli, F.; Scotti, G.; Olivieri, M. Klessydra-T: Designing Vector Coprocessors for Multithreaded Edge-Computing Cores. *IEEE Micro* **2021**, *41*, 64–71. [CrossRef]
12. Moghaddam, M.T.; Muccini, H. Fault-tolerant IoT. In Proceedings of the International Workshop on Software Engineering for Resilient Systems, Naples, Italy, 17 September 2019; Springer: Cham, Switzerland, 2019; pp. 67–84. [CrossRef]

13. Power, A.; Kotonya, G. A Microservices Architecture for Reactive and Proactive Fault Tolerance in IoT Systems. In Proceedings of the 2018 IEEE 19th International Symposium on "A World of Wireless, Mobile and Multimedia Networks" (WoWMoM), Chania, Greece, 12–15 June 2018; pp. 588–599. [CrossRef]
14. Ibrahim, M.; Baloch, N.K.; Anjum, S.; Zikria, Y.B.; Kim, S.W. An energy efficient and low overhead fault mitigation technique for internet of thing edge devices reliable on-chip communication. *Softw. Pract. Exp.* **2021**, *51*, 2393–2410. [CrossRef]
15. Zielinski, Z.; Wrona, K.; Furtak, J.; Chudzikiewicz, J. Reliability and Fault Tolerance Solutions for MIoT. *IEEE Commun. Mag.* **2021**, *59*, 36–42. [CrossRef]
16. Bertoa, T.G.; Gambardella, G.; Fraser, N.J.; Blott, M.; McAllister, J. Fault Tolerant Neural Network Accelerators with Selective TMR. *IEEE Des. Test* **2022**, *40*, 67–74. [CrossRef]
17. Tuli, S.; Casale, G.; Jennings, N.R. PreGAN: Preemptive Migration Prediction Network for Proactive Fault-Tolerant Edge Computing. In Proceedings of the IEEE INFOCOM, Online, 2–5 May 2022; pp. 670–679. [CrossRef]
18. Dong, B.; Wang, Z.; Chen, W.; Chen, C.; Yang, Y.; Yu, Z. OR-ML: Enhancing Reliability for Machine Learning Accelerator with Opportunistic Redundancy. In Proceedings of the 2021 Design, Automation & Test in Europe Conference & Exhibition (DATE), Grenoble, France, 1–5 February 2021; pp. 739–742. [CrossRef]
19. Zhang, J.J.; Basu, K.; Garg, S. Fault-Tolerant Systolic Array Based Accelerators for Deep Neural Network Execution. *IEEE Des. Test* **2019**, *36*, 44–53. [CrossRef]
20. Zheng, Z.; Zhou, T.C.; Lyu, M.R.; King, I. Component Ranking for Fault-Tolerant Cloud Applications. *IEEE Trans. Serv. Comput.* **2012**, *5*, 540–550. [CrossRef]
21. Javed, A.; Heljanko, K.; Buda, A.; Framling, K. CEFIoT: A fault-tolerant IoT architecture for edge and cloud. In Proceedings of the 2018 IEEE 4th World Forum on Internet of Things (WF-IoT), Singapore, 5–8 February 2018; pp. 813–818. [CrossRef]
22. Khan, W.Z.; Ahmed, E.; Hakak, S.; Yaqoob, I.; Ahmed, A. Edge computing: A survey. *Future Gener. Comput. Syst.* **2019**, *97*, 219–235. [CrossRef]
23. Rossi, D.; Conti, F.; Marongiu, A.; Pullini, A.; Loi, I.; Gautschi, M.; Tagliavini, G.; Capotondi, A.; Flatresse, P.; Benini, L. PULP: A parallel ultra low power platform for next generation IoT applications. In Proceedings of the 2015 IEEE Hot Chips 27 Symposium (HCS), Cupertino, CA, USA, 22–25 August 2015; pp. 1–39. [CrossRef]
24. Barbirotta, M.; Mastrandrea, A.; Menichelli, F.; Vigli, F.; Blasi, L.; Cheikh, A.; Sordillo, S.; Gennaro, F.D.; Olivieri, M. Fault resilience analysis of a RISC-V microprocessor design through a dedicated UVM environment. In Proceedings of the 33rd IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems, DFT 2020, Frascati, Italy, 19–21 October 2020. [CrossRef]
25. George, N.; Elks, C.R.; Johnson, B.W.; Lach, J. Transient fault models and AVF estimation revisited. In Proceedings of the 2010 IEEE/IFIP International Conference on Dependable Systems & Networks (DSN), Chicago, IL, USA, 28 June–1 July 2010; pp. 477–486. [CrossRef]
26. Waterman, A.; Lee, Y.; Patterson, D.A.; Asanovi, K. *The RISC-V Instruction Set Manual. Volume 1: User-Level ISA, Version 2.0*; Technical Report; Department of Electrical Engineering and Computer Sciences, California University Berkeley: Berkeley, CA, USA, 2014.
27. Aranda, L.A.; Wessman, N.J.; Santos, L.; Sánchez-Macián, A.; Andersson, J.; Weigand, R.; Maestro, J.A. Analysis of the critical bits of a RISC-V processor implemented in an SRAM-based FPGA for space applications. *Electronics* **2020**, *9*, 175. [CrossRef]
28. Wilson, A.E.; Wirthlin, M. Neutron radiation testing of fault tolerant RISC-V soft processor on Xilinx SRAM-based FPGAs. In Proceedings of the 2019 IEEE Space Computing Conference (SCC), Pasadena, CA, USA, 30 July–1 August 2019; pp. 25–32.
29. Ramos, A.; Toral, R.G.; Reviriego, P.; Maestro, J.A. An ALU protection methodology for soft processors on SRAM-based FPGAs. *IEEE Trans. Comput.* **2019**, *68*, 1404–1410. [CrossRef]
30. Santos, D.A.; Luza, L.M.; Dilillo, L.; Zeferino, C.A.; Melo, D.R. Reliability analysis of a fault-tolerant RISC-V system-on-chip. *Microelectron. Reliab.* **2021**, *125*, 114346. [CrossRef]