Journal of
Low Power Electronics and Applications

Article

Evaluation of Dynamic Triple Modular Redundancy in an Interleaved-Multi-Threading RISC-V Core

Marcello Barbirotta [1,*,†], Abdallah Cheikh [1,†], Antonio Mastrandrea [1,†], Francesco Menichelli [1,†], Marco Ottavi [2,3,†] and Mauro Olivieri [1,*,†]

1 Department of Information Engineering, Electronics and Telecommunications (DIET), "La Sapienza" University of Rome, Via Eudossiana 18, 00184 Rome, Italy
2 Department of Electronic Engineering, "Tor Vergata" University, Via del Politecnico 1, 00133 Rome, Italy
3 Faculty of Electrical Engineering, Mathematics and Computer Science, "University of Twente", Drienerlolaan 5, 7522 Enschede, The Netherlands
* Correspondence: marcello.barbirotta@uniroma1.it (M.B.); mauro.olivieri@uniroma1.it (M.O.); Tel.: +39-06-4458-5557 (M.O.)
† These authors contributed equally to this work.

Abstract: Functional safety is a key requirement in several application domains in which microprocessors are an essential part. A number of redundancy techniques have been developed with the common purpose of protecting circuits against single event upset (SEU) faults. In microprocessors, functional redundancy may be achieved through multi-core or simultaneous-multi-threading architectures, with techniques that are broadly classifiable as Double Modular Redundancy (DMR) and Triple Modular Redundancy (TMR), involving the duplication or triplication of architecture units, respectively. RISC-V plays an interesting role in this context for its inherent extendability and the availability of open-source microarchitecture designs. In this work, we present a novel way to exploit the advantages of both DMR and TMR techniques in an Interleaved-Multi-Threading (IMT) microprocessor architecture, leveraging its replicated threads for redundancy, and obtaining a system that can dynamically switch from DMR to TMR in the case of faults. We demonstrated the approach for a specific family of RISC-V cores, modifying the microarchitecture and proving its effectiveness with an extensive RTL fault-injection simulation campaign.

Keywords: fault-tolerance; fault-detection; fault-injection; microprocessors; RISC-V; Interleaved-Multi-Threading (IMT)

check for updates

Citation: 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.* **2023**, *13*, 2. https://doi.org/10.3390/jlpea13010002

Academic Editors: Teresa Cervero, Kevin Martin, Mario Kovač and Maurizio Martina

Received: 4 October 2022
Revised: 19 December 2022
Accepted: 23 December 2022
Published: 28 December 2022

1. Introduction

In the last few decades, the growing complexity of electronic systems employed in the automotive, aerospace, military, and generally in safety-critical applications has increased the importance of fault-tolerant design and fault simulation. Moreover, the probability of faults in digital electronic devices has increased with technology scaling, voltage margin reduction, and statistical process variations magnification [1–3]. As a consequence, fault-tolerance (FT) techniques for functional safety support in microprocessor cores have become a requisite in many system designs. It is widely known that the FT challenge is best approached through the application of different redundancy techniques, often placed at different abstraction levels: software [4], system architecture, microarchitecture, and single functional units [5]. RISC-V has recently gained attention for safety applications, thanks to its extendable instruction set and its many open-source implementations that allow for the exploration of multiple techniques to achieve FT [6–8]. On one hand, Triple Modular Redundancy for some fault models, regardless of the abstraction level at which it is inserted (i.e., registers, units, or system), could have the advantage of a virtually immediate error correction action, at the expense of a triplication of hardware resources or of the execution time. On the other hand, Double Modular Redundancy intrinsically has a lower overhead, but it usually relies on recovering the correct state of the architecture using a relatively expensive procedure. DMR just allows for error detection, since voting techniques require three units to detect and to correct an error: in the case of a mismatch, some kind of higher-level handler (typically a software exception handler) is called to restore a previously saved safe state. Periodically saving the safe state (checkpoint) intrinsically has a cost in terms of performance. As a consequence, conventional DMR implementations have a cost that is more than two times the non-redundant system. Yet, considering a typical space application with SEU events in the order of one per day [9], a classical TMR scheme may be too expensive to protect the architecture, while applying a DMR solution could be the easiest choice in terms of power consumption and hardware overhead, with the only drawback being the implementation of some checkpoint and restore methodologies to save the last correct state of the core and restore it in case of mismatches in the DMR logic due to faults.

This work is centered on the application of the DMR paradigm within an Interleaved-Multi-Threading (IMT) RISC-V architecture, gaining the low overhead advantages of the DMR technique, and yet overcoming the cost of saving checkpoints and restoring the software state using *Dynamic TMR* (DTMR) protection, which actually implies the behavior of a TMR only in the case of error detection.

The contributions of the proposed work are:

- To demonstrate that thanks to the inherent behavior of an Interleaved-Multi-Threading structure, the use of restoring mechanisms through checkpointing routines is unnecessary and can be replaced by a *Dynamic TMR* mechanism;
- To demonstrate the concept of *Dynamic TMR* and how it can be applied to an existing RISC-V IMT core;
- To report the evaluation of the effectiveness of the proposed technique in a RISC-V IMT core through an extensive fault-injection (FI) simulation.

The rest of the paper is organized as follows: Section 2 discusses the related works on DMR approaches and multi-thread fault-tolerant processors, Section 3 outlines the proposed methodology, Section 4 details the implementation of the methodology in a critical unit of the processor core, Section 5 describes the validation setup, Section 6 reports the experimental results, and Section 7 summarizes our conclusions.

2. Related Works

A considerable number of redundancy techniques at the processor architecture level have been developed over the past decades. These methods span from software approaches with instruction redundancy [10–12] to multi-threading systems that have been realized via Simultaneous Multi-Threading (SMT) [13–15] and Multi-Core (MC) approaches [16–18]. In those systems, multiple logical or physical cores run the same computation at the instruction or task level, thus obtaining FT through modular redundancy [19]. Some works try to find ways to increase the reliability of basic TMR or DMR approaches. In [20], a DMR with global/partial reservation is introduced and compared in terms of reliability with TMR. DMR with global reservation presents a duplicated DMR structure with a total of four identical modules. When one of the first pair fails, the pair is replaced at runtime with the other pair. Similarly, the DMR with partial reservation has four identical units, and when one of the two units fails, only the failed unit is replaced by a reserved unit. The work concludes that TMR is more reliable than global reservation DMR, but that it is less reliable than partial reservation DMR, which is in any case more expensive in terms of hardware resources.

Other works try to mitigate the overhead in performance, power consumption, or the hardware of TMR and DMR. In [21,22], we presented the advantages of combining spatial and temporal redundancy techniques in an Interleaved Multi-Threading (IMT) RISC-V core, implementing a *Buffered Triple Modular Redundancy* (BTMR) with limited hardware costs and good results in terms of fault-tolerance. In [23], we explored the extension of the same approach to a fault-tolerant vector coprocessor, targeting high performance edge computing applications. Other studies on DMR techniques show that it allows significant savings with respect to TMR, especially considering the benefits regarding the area overhead and power consumption. In [24], the authors use DMR and introduce design diversity between two modules to produce different error patterns and easily detectable mismatches, yet still relying on checkpointing and restoring. The authors of [25] compare three different checkpointing schemes for DMR multi-core architectures, Store-Compare-Checkpointing (CSCP), Store-Checkpointing (SCP), and Compare-Checkpointing (CCP), finding the optimal Checkpointing number to minimize the Mean Execution Time of a single task. In [26], a novel Complementary DMR (CDMR) technique is described, inspired by the Markov Random Field (MRF) theory, with two voting stages being optimized with MRF and exhibiting a high performance in merging the results, obtaining better values in terms of the performance overhead, and fault resilience in the voting logic. Introducing redundancy is not always necessary: The authors in [27] present a way to perform sampling-DMR, enabling DMR for only a small fraction of time (1% of a 5 million cycles time slot), gaining in power consumption and design complexity. DMR can be applied at different architectural levels and it can be activated only in the case of need, through runtime Architectural Vulnerability Factor (AVF) estimation [28]. The authors of [29] present an FPGA-CPU architecture with a self-monitoring scheme and health indicators, where parameters such as temperature, usage, etc., are monitored to obtain the health state for both systems, choosing the best one and performing software voting when the results are available.

The authors in [30] build a DMR system combining a lock-step technique for error detection and checkpointing, with rollback for error recovery. They use a checker logic block to detect the errors and to periodically issue an interrupt request for a checkpoint using a DMA transfer. In [31], similar to [30], DMR is realized through a dual-core lock-step architecture, using two CPUs in a ARM Cortex-A9 processor, running FreeRTOS and using interrupts to handle the checkpoint operations and rollback. When the CPU raises an interrupt, the actual thread is stalled and the processor registers are saved in the stack [31]. Another solution is described in [32], where a 666 MHz Arm A9 hard-core and a 25 MHz LowRISC soft-core on Zynq-7000 is run in a dual-core lock-step configuration. The system can stop, restart, restore from a checkpoint, or continue execution. In [33], the authors present CEVERO, a RISC-V System-on-Chip that implements FT on a PULP platform [34–36]. It comprises two Ibex cores running in a lock-step configuration, which are monitored via an FT hardware module that checks whether an error occurs for every executed instruction. The lock-step technique, combined with rollback recovery and checkpoints, for cores synthesized in an FPGA, was first introduced by [37] using a Virtex II-Pro.

3. Proposed Approach

3.1. Klessydra-fT03 Microarchitecture

Klessydra-fT03 [21,22] is a fault-tolerant RISC-V processor core that uses an Interleaved-Multi-Threading (IMT) architecture as a basis for the implementation of a radiation hardening technique called *Buffered TMR*. Klessydra-fT03 is based on an open-source RISC-V softcore family, namely Klessydra-T, which interleaves three or more hardware threads in a round-robin fashion on a four-stage in-order pipeline that is fully compatible with the PULPino open-source microcontroller platform [34,38].

The basic concept of Klessydra-fT03 is the intrinsic FT capability of an IMT core running three identical threads, each having its own Register File (RF), Program Counter (PC), and Control/Status Registers (CSRs), thus incorporating *spatial redundancy*, yet sharing the pipeline logic and registers to execute the same instructions at different clock cycles in order to vote on the results, thus providing *temporal redundancy* without adding additional main memory locations to save the data that are produced by the redundant instructions. Differently from the FT approaches based on Simultaneous Multi-Threading (SMT) or Multi-Cores (MC) schemes [15,18,20], the technique exploits IMT by introducing specific logic structures to deal with the fact that the instruction results are not simultaneously available.

Executing the same instructions in duplicated threads on the same hardware at different clock cycles protects the architecture from Single Event Upset (SEU) in sequential logic and from Single Event Transient faults (SET) that may occur in combinational logic. The *Buffered TMR* paradigm defines precise architectural modifications with general validity. The values produced by the three hardware threads in specific architectural units (PC, Register File, Write Back unit, and Load Store Unit) are *buffered* in dedicated registers and voted at the end of each instruction, thus performing an intrinsic TMR protection and correct data retention.

3.2. The Dynamic TMR Principle

The principle of the *Dynamic TMR* is implemented in a new core that we named Klessydra-dfT03 (microarchitecture in Figure 1), where "d" stands for dynamic. It builds on the idea of turning the *Buffered TMR* into a DMR technique, to reduce the power consumption and to increase the speed. In fact, leveraging the multi-threaded architecture to use only two replicated threads instead of three, we ideally save 1/3 of the execution time and dynamic energy consumption. Yet, a checkpoint and restore mechanism causes the re-execution of many instructions that have already been completed correctly, and it also introduces hardware overheads to store checkpoints, and time overheads to execute the software routines for the restoring procedures. So, we exploited the IMT operation to retrieve the correct state as the one corresponding to the instruction preceding the clock cycle in which the fault was detected. This is achieved by introducing a single register that saves the address of the last correct instruction and restores it in the PC with a latency overhead of four clock cycles, without any changes in the data memory or the Register File. A fundamental aspect of the approach is the capability of maintaining the same fault resilience performances of a *Buffered TMR* system.

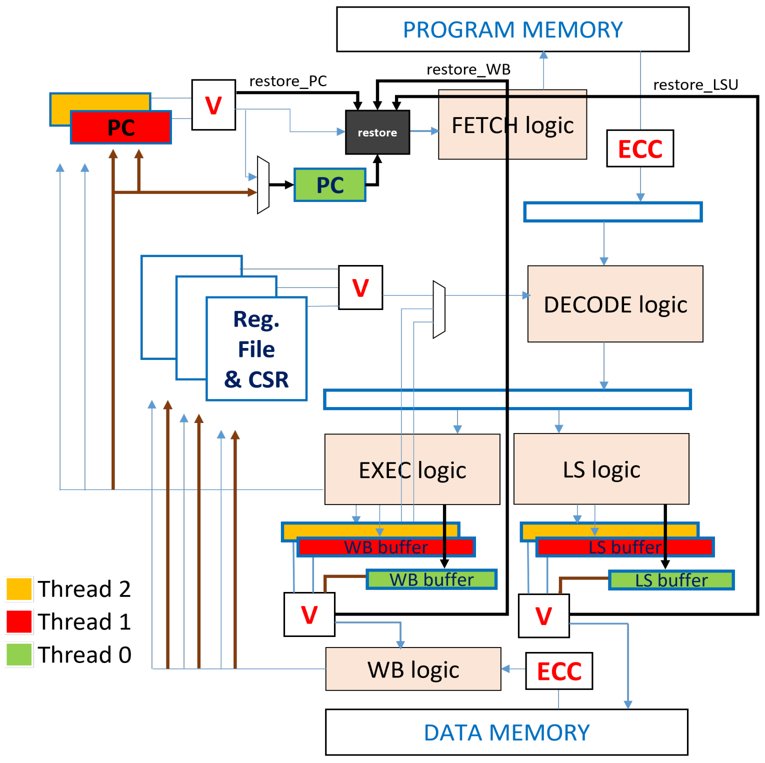

Figure 1. Klessydra-dfT03 microarchitecture. Blue arrows: Normal mode; black arrows: Restore mode; brown arrows: End Restore Phase.

The feature at the basis of the methodology is the ability to dynamically turn threads on and off within a RISC-V Klessydra core. We use three hardware threads, numbered as Thread 2, Thread 1, and Thread 0 (blue, red, and green colors in Figure 1), and we leave only the threads 2 and 1 active while turning off Thread 0, which we call *the auxiliary thread* that is activated only in case of fault detection. The auxiliary thread does not take part in the pipeline normal operations, it does not fetch any instruction, and its dedicated hardware units (PC, Registers, and Control-Status Registers), although present, are inactive and they therefore have no dynamic consumption. The operation of the processor is organized into three operating modes:

- **Normal or "*Buffered DMR*" mode:** Threads 2 and 1 work in interleaved mode (blue arrows in Figure 1), executing the same instructions and thus implementing spatial and temporal redundancy, with a *buffered* voting mechanism implemented in the critical units PC, Register File, Write Back unit, and Load Store Unit, that check for the correctness of the program execution.
- **Restore or Recovery mode:** If the voting logic gives a negative result due to a fault, specific control signals named *restore_* signals (Figure 1) are asserted, and the core enters the recovery mode. Notably, a fault is always detected before the Register File would be updated with a wrong result using the faulted instruction. Following the black arrows in Figure 1, the *restore_* signals activate the restore block (black unit in Figure 1), which wakes up the sleeping auxiliary thread. As the new thread enters the IMT pipeline, it fetches the last successfully executed instruction indicated by the dummy PC register (see next section), while the other threads are stalled.
- **End of Restore Phase:** Once the recovered instruction is completed, the produced result is compared with the results previously produced by the other two mismatching threads (brown arrows in Figure 1), thus obtaining a majority voting similar to a TMR system, and writing back the correct value into the Register File. The recovery procedure ends with the suspension of the auxiliary Thread 0, and the loading of the address of the next instruction in the PCs of Threads 2 and 1, so that they restart from the instruction following the one that faulted.

Thanks to this technique, it is possible to obtain all of the characteristics of a *Buffered TMR* system, gaining in terms of speed and energy. We refer to the above described mechanism as *Dynamic TMR*, as we can dynamically add TMR redundancy only when a fault occurs.

4. An Example of Implementation and Operation

As mentioned in the previous section, in an IMT pipeline, there are several critical points in the architecture to perform voting in order to reveal the presence of faults and to activate the recovery mode. We implemented *checking units* in the PC, Register File, Write Back unit, and Load Store Unit of the microarchitecture, for which we report the FT performance results in Section 6. For the sake of conciseness, here we report the details of the PC unit design. The PC is the most critical unit for the Restore mode as it contains the logic providing the checkpoint of the correct instruction.

In an IMT structure, each Thread has its own Program Counter. In the dfT03 microarchitecture, the PC of the auxiliary Thread 0 is used as a dummy register to save the last correct instruction address to restart in the case where a fault is detected.

During the normal mode of operation, executing one instruction per clock cycle, the PC belonging to Thread 0 is updated every two clock cycles, with the correct value coming from the voting between the PC of Thread 1 and Thread 2, respectively, in the Decode and Execute stage. Figure 2 represents the temporal flow of the instructions (distinguished by the different colors) in the different threads within the pipeline. A fault inside the Program Counter unit could lead to unwanted jumps or invalid instructions. For this reason, if a fault hits a thread with a consequent change of the address value inside the Program Counter, the voting procedure manages to detect the mismatch between the two PC registers and it raises a signal that is capable of starting the Restore mode. As can be seen in Figure 2, once the restore procedure starts, the PC of Thread 0 already contains the address of the last correctly executed instruction (green color in the figure); thus, the instruction is fetched and inserted in the pipeline, and it is executed again. Assuming that there are no further faults during the restore procedure (which is by far a statistically realistic assumption according to the typical fault rates [9]), the execution of the last correct instruction produces the valid address of the next instruction, contained in the Thread 0 PC. The address is loaded in the PCs of Thread 2 and Thread 1, so they can perform a fetch of the next correct instruction, and they resume the regular operation. As a proof of concept, in Figure 3, we report the waveforms referring to a restore procedure for the PC unit, activated by a fault. It is possible to distinguish the various phases described above, such as the awakening of Thread 0 due to the fault on the signal *pc_ID*, which changes the instruction address from 0×714 to 0×734. The restore procedure starts with the *restore_fault_PC_signal* and the previous instruction at address 0×710 is executed again; then the core fetches the next instruction at address 0×714, for both the remaining threads.

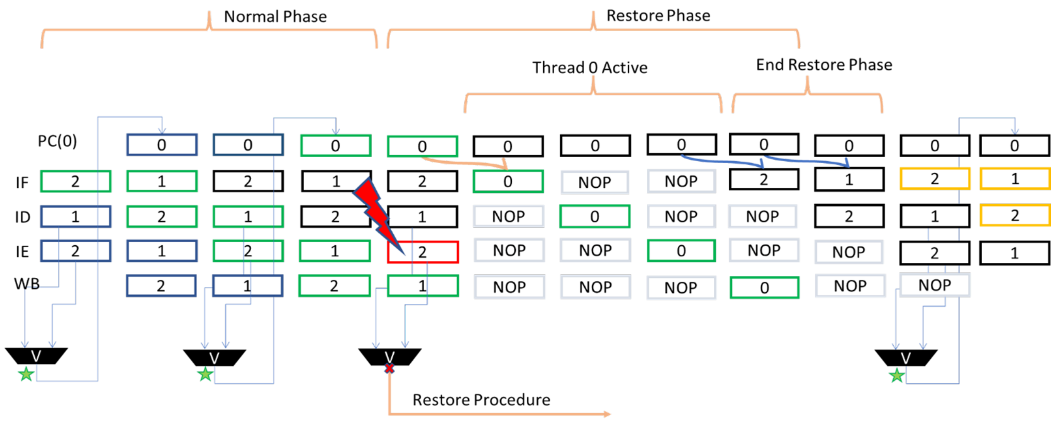

Figure 2. Restore Phase for Program Counter unit. The green stars represent voting performed correctly, while the red x represents a detected error

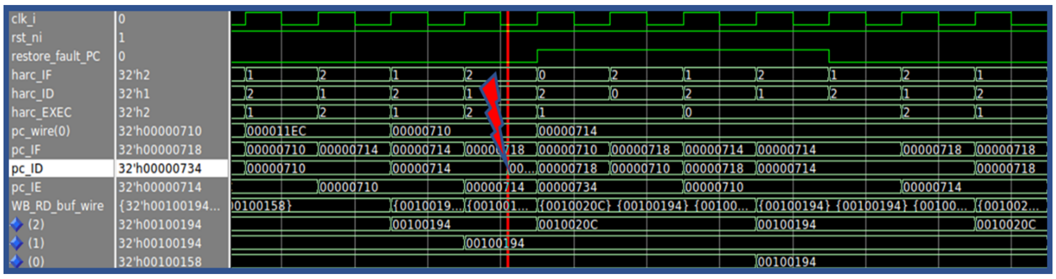

Figure 3. Restore Phase example for Program Counter unit.

5. Validation Setup

The setup used for FI analysis is based on the Time Frame Spanning method [39], which targets all of the synchronous register bits in the microarchitecture under analysis, while simulating the execution of a set of software test programs on the microarchitecture.

5.1. The Time Frame Span Approach

Unlike the statistical Monte Carlo FI methods, the adopted method consists of the deterministic injection of faults into the architecture, within a time interval for each target bit, during the application program execution simulation. The technique divides the whole execution time into m subintervals called *time frames*. For each time frame and for each ACE bit in the microarchitecture, a simulation is run, injecting bit flips on the target bit at intervals of a very few clock cycles. The methodology identifies whether a bit is ACE for a specific program to be executed. The approach is well suited to relatively simple programs such as repetitive computation kernels, such as in the case of many microcontroller applications. The analysis ends when all of the ACE bits have been tested in all of the time frames, totaling $m \times n$ simulation runs, with n being the number of ACE bits to be targeted.

Supposing the fault simulation analysis targeting the j-th bit of the microarchitecture reports system failures in only mf frames out of the total m time frames, we know the deterministic information that in $(m - mf)$ frames, the execution does not fail even if a fault hits the target bit in that time interval, so that assuming a uniform time distribution of SEU events (a statistically realistic assumption according to the typical SEU fault rates [9]), we can state:

$$\mathrm{Pf}(j) \triangleq \mathrm{Pr}\Big\{\text{a SEU hitting } j^{\text{th}} \text{ bit causes a system fault}\Big\} \leq m_F/m \quad (1)$$

The obtained Pf(j) represents a lower bound value of the resilience of the core with respect to the faults on the the j-th bit when running the specific software application program. In other words, it identifies the time frames during program execution in which the system (hardware + software) is immune to faults on the j-th bit. For example, if 6 out of 10 time-frame simulations result in a failure for a certain bit, the obtained probability of failure due to SEUs on that bit is below 60%. By performing a sub-division of failure-affected frames into smaller frames, it is possible to further increase the accuracy grade of the obtained probability, which depends not only on the number of injected errors, but is also proportional to the number of time frames.

This method is particularly valid in space applications where the SEU rate can be in the order of 1 per day, and the embedded processor repeats the application program routine continuously [9]. The event can damage any bit of the architecture with the same probability since the execution of the program routine and the occurrence of a physical event are uncorrelated.

In the baseline T03 microarchitecture (having no fault-tolerance mechanism), the Time Frame Spanning fault simulation campaign targets all of the register bits. In addition to characterizing the failure probability, the analysis also identifies the bits that may actually cause a program failure when faulted. In the literature, such bits are referred to as the Architecturally Correct Execution (ACE) bits associated with the program being run. By definition, non-ACE bits do not produce a program failure when faulted. When doing the Time Frame Spanning fault simulation campaign on the hardened microarchitecture (e.g., fT03 and dfT03), injecting faults on non-ACE bits would produce no failure anyway: therefore, the simulation time is optimized by avoiding injecting faults on non-ACE bits [21,40].

As we mentioned in the previous section, we assume that during the restore mode of the processor following a fault, no additional faults occur in the microarchitecture. In the statistically unlikely case that a second fault occurs, the system will fail. Thus, in our fault-injection simulation analysis, the injection of bit flips inside a DMR core is disabled during the restore procedure. The presented analysis is limited to SEU effects to demonstrate the potential of the DTMR technique with respect to other FT approaches. For a space-qualification of a full product based on the proposed microarchitecture, other effects of perturbations should also be modeled and analyzed. The underlying assumption made to estimate the failure probability, based on the results of the fault-injection characterization, is that physical SEU faults occur with a uniform random distribution in space and time, with respect to the duration of a program execution on the processor. The approach is detailed in [39]. The concepts of error margin and confidence level are not applicable to the proposed approach because it is not based on testing a statistical sample of the population of the bits, like in a Monte Carlo approach, but on testing the whole population of the bits of the microarchitecture. In the view of a characterization of the risk according to standards such as ISO 26262, which are beyond the scope of the reported research, the error margin may be studied with respect to the time resolution of the fault-injection campaign. For the purpose of the proposed research, we limit our analysis to give an upper bound of the failure probability.

5.2. Test Programs

We analyzed the DTMR dfT03 core running specific computational kernel benchmarks, representative of typical code executed in embedded numerical applications: FFT, CRC32, and the FIR filter. A summary of the benchmark setup is shown in Table 1. In order to observe the improvements offered by the proposed approach, we compared the Klessydra-dfT03 core with the non-redundant three-thread Klessydra-T03 core, in which a single thread ran the benchmark while the others were set idle (by executing a WFI—Wait For Interrupt RISC-V instruction). As an additional case, we compared the new architecture with the fault-tolerant Klessydra-fT03 core [21,22], where redundancy is obtained with the BTMR technique. Looking at the total clock cycles shown in Table 1, it is possible to note that completing the same program in the dfT03 core requires about 2/3 of the total cycles with respect to fT03; this is due to the fact that dfT03 works as a DMR core in absence of faults. As for the T03 core, maintaining two threads in a WFI state allows for program completion in slightly less cycles with respect to fT03.

Table 1. Test setup with clock cycles required to complete the benchmarks; number of frames used by the Time Frame Spanning techniques [39] and the total faults per frame, with an average of 1 fault every 35 clock cycles.

	fft			*crc32*			*fir*		
core	**dfT03**	**fT03**	**T03**	**dfT03**	**fT03**	**T03**	**dfT03**	**fT03**	**T03**
Total clock cycles	106,090	159,492	134,192	15,042	20,037	18,563	49,140	72,566	64,653
# frames	10			10			10		
faults / frame	250	425	355	40	53	50	131	194	170
Deterministic fault rate	1 every 35 cycles			1 every 35 cycles			1 every 35 cycles		

6. Experimental Results

The FI test campaign led to the results summarized in Figure 4. The diagram shows the probability of failure (Pf) regarding the non-protected T03 core (red bars) and the protected fT03 and dfT03 cores (light and dark green bars) for all the target registers (registers containing ACE bits). For registers larger than 1-bit, we plotted the arithmetic average of all its bits.

The non-protected architecture experiences failure rates of close to 100% in most registers. It is possible to see when observing the dfT03 bars that for all the benchmarks there has been a significant reduction in failures in almost all of the signals. It is possible to note that the failures completely disappear in many registers, even for relatively large bit-widths (e.g., 32-bit). A particular outcome regards the highest failing ACE register in the protected dfT03 architecture, the *LS_WB* register, which contains the value read from the load-store unit in the case of a load operation. The register cannot be *buffered* to apply the DMR check, because in the adopted method, the load operation is performed only once by the already buffered control signals and the data from LSU. Even if the memory bus is ECC-protected, so the value loaded into *LS_WB* can be considered correct; when a fault occurs in the register, there is no way to protect it since its value arrives at the end of the *buffered* load operation [21,22]. In this case, a standard TMR mechanism to protect the register should be the preferred choice. The other registers with non-zero failure probability in the dfT03 core are just three, and they all stay below 2% Pf. Comparing the two redundant architectures, it is possible to note that the dfT03 core has a higher fault-tolerance than the fT03 core in the case of single-event failures.

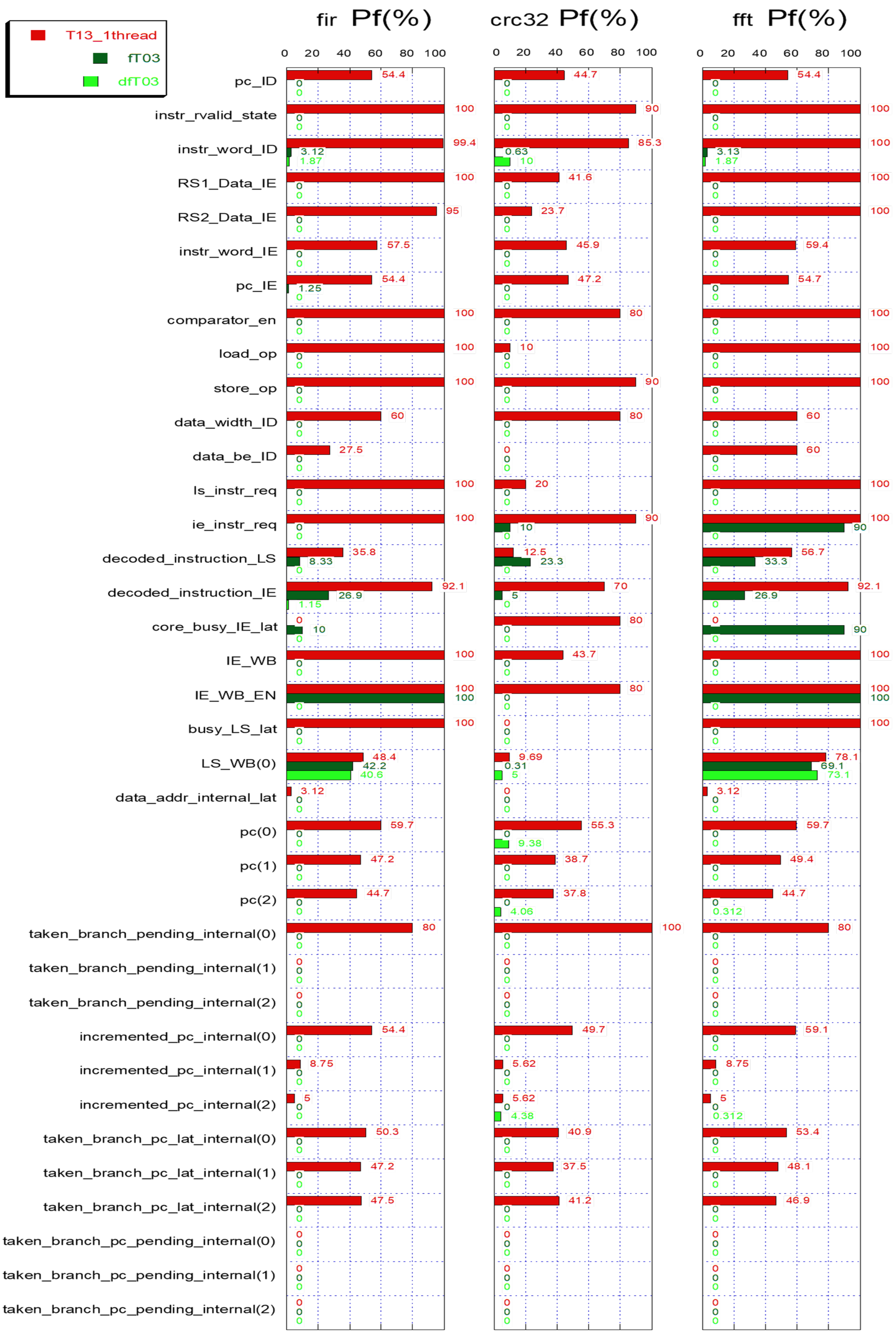

Figure 4. Fault-injection results with a Time Frame Spanning approach for the T03, fT03, and dfT03 cores, running the FFT, CRC32, and FIR benchmarks. Target registers on the vertical axis and failure probability (%) on the horizontal axis.

7. Performance Comparison Analysis

In order to clarify the gain in terms of execution time, it is possible to compare the performance of our core with MC DMR + checkpointing architectures. In [30,31], the checkpoint and the restore routines are implemented as a software interrupt handler. In the fault-free case, only the checkpoint routine affects the overall execution time, depending on the checkpointing interval period. In the same case, the dfT03 core does not exhibit a time overhead for the checkpoint operation, as this is automatically performed by the hardware architecture, saving on a dedicated register the last correct program counter for every executed instruction. As an example, authors in [30] chose to perform a checkpoint every 300 write cycles, obtaining a time overhead ranging from 17% for a FIR benchmark to 54% for a Kalman filtering benchmark. Similarly, the authors in [31] compare a matrix multiplication benchmark executed as bare-metal, and in a FreeRTOS environment, showing an overhead ranging from 59% for the bare-metal case up to 81% for the FreeRTOS case.

In the presence of faults, the restore routine additionally affects the program execution time depending on the fault rate, making direct comparisons more difficult. The mean recovery time for the dfT03 architecture is four clock cycles, as the affected instruction must be reintroduced into the pipeline, starting from the fetch stage. The others approaches require the restoration of the entire register file, in addition to the time spent on the interrupt routine. The recovery time of dfT03 is then smaller than any other technique using software rollback routines.

From the fault-tolerance point of view, it is not very meaningful to directly compare different FT cores, unless they are subject to the same FI tests. However, it is possible to observe the quality of a protection technique by evaluating the percentage of mitigated faults with respect to the total injected faults. In [37] the work presents a fault mitigation of around 54%, while in [30], the mitigation value is 84%. In [31], the authors report a range from 60% to 70% for the bare-metal environment, and a range from 50% up to 60% for the FreeRTOS environment. The works in [32,33] do not report the number of injected faults, while [33] refers to FI focused on protected units only, claiming that the system can detect errors, but no numerical data about FT performance and fault coverage are shown, except for the recovery time overhead, which is reported as being 40 cycles long. In the present work, by averaging the results in Figure 4, the error mitigation rate ranges between 96.8% and 98.6% for the ACE bits in the architecture on the tested benchmarks.

8. Impact on Hardware Resources

All the architectures have been synthesized using Xilinx Vivado 2019 on a Genesys2 board, based on Xilinx Kintex-7 FPGA. Table 2 shows the hardware resource utilization. All of the designs have been constrained to a 107 MHz clock frequency, which is the maximum frequency for the Klessydra-dfT03 core. It is possible to see that the proposed design, compared to fT03, has a slight increase in the number of LUTs and FFs, due to the thread recovery logic. Comparing dfT03 to the original T03 version, the increase in hardware resource requirements is justified by the introduction of fault-tolerance features.

Table 2. Synthesis results using Xilinx Vivado 2019 on a Genesys2 board (Kintex-7 FPGA).

Core	LUTs	FFs	Energy [pJ/cycle]
T03 (non-hardened)	5524	4489	380
fT03 (hardened)	6429	4905	390
dfT03 (hardened)	6923	5019	390

9. Conclusions

In this work, we started from an open-source RISC-V microprocessor core design to implement and to evaluate the new hardening-by-design DTMR technique conceived for the IMT RISC-V cores, demonstrating its resilience to SEU faults through an extensive fault-injection simulation analysis. We compared the new DTMR design to the non-protected version of the core, and the previous BTMR technique from the fault-tolerant point of view and hardware cost. We observed that the proposed DTMR design has better fault-tolerance levels at the cost of a slight increase in hardware resources with respect to the BTMR core. The DTMR technique relies on dynamically waking up an auxiliary thread only in the case of fault detection in the two active redundant threads, allowing us to recover the correct state of the machine without using any checkpoint/restore software procedure, over an average time of four clock cycles. The FI simulation was based on a Time Frame Spanning deterministic simulation, achieving a full coverage of the microarchitecture registers, unlike the Monte Carlo statistical FI, and the results showed a drastic reduction in the system failure probability when running numerical processing software benchmarks. The technique opens the way to further investigations to extend the approach to more complex architectures composed of heterogeneous cores and accelerators.

Author Contributions: Conceptualization, M.B., M.O. (Mauro Olivieri) and M.O. (Marco Ottavi); methodology, M.B.; software, M.B.; validation, A.C. and A.M.; formal analysis, M.O. (Mauro Olivieri) and F.M.; investigation, M.B. and M.O. (Marco Ottavi); resources, M.B. and A.M.; data curation, M.B. and A.C.; writing—original draft preparation, M.B.; writing—review and editing, F.M., M.O. (Marco Ottavi) and M.O. (Mauro Olivieri); visualization, M.B. and F.M.; supervision, M.O. (Mauro Olivieri); project administration, M.O. (Mauro Olivieri); funding acquisition, M.O. (Marco Ottavi). All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

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

References

1. Azimi, S.; Sterpone, L. Digital design techniques for dependable high performance computing. In Proceedings of the 2020 IEEE International Test Conference (ITC), Washington, DC, USA, 1–6 November 2020; pp. 1–10.
2. De Sio, C.; Azimi, S.; Sterpone, L.; Du, B. Analyzing Radiation-Induced Transient Errors on SRAM-Based FPGAs by Propagation of Broadening Effect. *IEEE Access* **2019**, *7*, 140182–140189. [CrossRef]
3. Buzzin, A.; Rossi, A.; Giovine, E.; de Cesare, G.; Belfiore, N.P. Downsizing Effects on Micro and Nano Comb Drives. *Actuators* **2022**, *11*, 71. [CrossRef]
4. De Sio, C.; Azimi, S.; Portaluri, A.; Sterpone, L. SEU evaluation of hardened-by-replication software in RISC-V soft processor. 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. 1–6.
5. Azimi, S.; De Sio, C.; Sterpone, L. In-Circuit Mitigation Approach of Single Event Transients for 45nm Flip-Flops. In Proceedings of the 2020 IEEE 26th International Symposium on On-Line Testing and Robust System Design (IOLTS), Napoli, Italy, 13–15 July 2020; pp. 1–6.
6. Li, J.; Zhang, S.; Bao, C. DuckCore: A Fault-Tolerant Processor Core Architecture Based on the RISC-V ISA. *Electronics* **2021**, *11*, 122. [CrossRef]
7. 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]
8. 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.
9. Carmichael, C.; Fuller, E.; Fabula, J.; Lima, F. Proton testing of SEU mitigation methods for the Virtex FPGA. In Proceedings of the Military and Aerospace Applications of Programmable Logic Devices MAPLD, 2001.
10. Reis, G.A.; Chang, J.; August, D.I. Automatic instruction-level software-only recovery. *IEEE Micro* **2007**, *27*, 36–47. [CrossRef]
11. Reis, G.A.; Chang, J.; Vachharajani, N.; Rangan, R.; August, D.I. SWIFT: Software implemented fault tolerance. In Proceedings of the International Symposium on Code Generation and Optimization, San Jose, CA, USA, 20–23 March 2005; pp. 243–254.
12. Serrano-Cases, A.; Restrepo-Calle, F.; Cuenca-Asensi, S.; Martínez-Álvarez, A. Softerror mitigation for multi-core processors based on thread replication. In Proceedings of the 2019 IEEE Latin American Test Symposium (LATS), Santiago, Chile, 11–13 March 2019; pp. 1–5.
13. Ma, Y.; Zhou, H. Efficient transient-fault tolerance for multithreaded processors using dual-thread execution. In Proceedings of the 2006 International Conference on Computer Design, San Jose, CA, USA, 1–4 October 2006; pp. 120–126.
14. Sundaramoorthy, K.; Purser, Z.; Rotenberg, E. Slipstream processors: Improving both performance and fault tolerance. *ACM SIGPLAN Not.* **2000**, *35*, 257–268. [CrossRef]
15. Osinski, L.; Langer, T.; Mottok, J. A survey of fault tolerance approaches on different architecture levels. In Proceedings of the ARCS 2017; 30th International Conference on Architecture of Computing Systems, VDE, Vienna, Austria, 3–6 April 2017; pp. 1–9.

16. Shernta, S.A.; Tamtum, A.A. Using triple modular redundant (tmr) technique in critical systems operation. In Proceedings of the Proceedings of First Conference for Engineering Sciences and Technology (CEST-2018), Garaboulli, Libya, 25–27 September 2018; Volume 1, p. 53.
17. Gomaa, M.; Scarbrough, C.; Vijaykumar, T.; Pomeranz, I. Transient-fault recovery for chip multiprocessors. In Proceedings of the 30th Annual International Symposium on Computer Architecture, San Diego, CA, USA, 9–11 June 2003; pp. 98–109.
18. Oz, I.; Arslan, S. A survey on multithreading alternatives for soft error fault tolerance. *ACM Comput. Surv. (CSUR)* **2019**, *52*, 1–38. [CrossRef]
19. Vargas, V.; Ramos, P.; Méhaut, J.F.; Velazco, R. NMR-MPar: A fault-tolerance approach for multi-core and many-core processors. *Appl. Sci.* **2018**, *8*, 465. [CrossRef]
20. Popov, G.; Nenova, M.; Raynova, K. Reliability Investigation of TMR and DMR Systems with Global and Partial Reservation. In Proceedings of the 2018 Seventh Balkan Conference on Lighting (BalkanLight), Varna, Bulgaria, 20–22 September 2018; pp. 1–4.
21. 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.
22. 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]
23. 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, SU, Italy, 12–15 June 2022; pp. 237–240.
24. Reviriego, P.; Bleakley, C.J.; Maestro, J.A. Diverse double modular redundancy: A new direction for soft error detection and correction. *IEEE Des. Test Comput.* **2013**, *30*, 87–95. [CrossRef]
25. Nakagawa, S.; Fukumoto, S.; Ishii, N. Optimal checkpointing intervals of three error detection schemes by a double modular redundancy. *Math. Comput. Model.* **2003**, *38*, 1357–1363. [CrossRef]
26. Li, Y.; Li, Y.; Jie, H.; Hu, J.; Yang, F.; Zeng, X.; Cockburn, B.; Chen, J. Feedback-based low-power soft-error-tolerant design for dual-modular redundancy. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **2018**, *26*, 1585–1589. [CrossRef]
27. Nomura, S.; Sinclair, M.D.; Ho, C.H.; Govindaraju, V.; De Kruijf, M.; Sankaralingam, K. Sampling+ dmr: Practical and low-overhead permanent fault detection. *ACM SIGARCH Comput. Archit. News* **2011**, *39*, 201–212. [CrossRef]
28. Vadlamani, R.; Zhao, J.; Burleson, W.; Tessier, R. Multicore soft error rate stabilization using adaptive dual modular redundancy. In Proceedings of the 2010 Design, Automation & Test in Europe Conference & Exhibition (DATE 2010), Dresden, Germany, 8–12 March 2010; pp. 27–32.
29. Matsuo, I.B.M.; Zhao, L.; Lee, W.J. A dual modular redundancy scheme for CPU–FPGA platform-based systems. *IEEE Trans. Ind. Appl.* **2018**, *54*, 5621–5629. [CrossRef]
30. Violante, M.; Meinhardt, C.; Reis, R.; Reorda, M.S. A low-cost solution for deploying processor cores in harsh environments. *IEEE Trans. Ind. Electron.* **2011**, *58*, 2617–2626. [CrossRef]
31. de Oliveira, Á.B.; Rodrigues, G.S.; Kastensmidt, F.L. Analyzing lockstep dual-core ARM cortex-A9 soft error mitigation in FreeRTOS applications. In Proceedings of the Proceedings of the 30th Symposium on Integrated Circuits and Systems Design: Chip on the Sands, Fortaleza, Brazil, 28 August–1 September 2017; pp. 84–89.
32. Rodrigues, C.; Marques, I.; Pinto, S.; Gomes, T.; Tavares, A. Towards a Heterogeneous Fault-Tolerance Architecture based on Arm and RISC-V Processors. In Proceedings of the IECON 2019-45th Annual Conference of the IEEE Industrial Electronics Society, Lisbon, Portugal, 14–17 October 2019; Volume 1, pp. 3112–3117.
33. Silva, I.; do Espírito Santo, O.; do Nascimento, D.; Xavier-de Souza, S. Cevero: A soft-error hardened soc for aerospace applications. In *Proceedings of the Anais Estendidos do X Simpósio Brasileiro de Engenharia de Sistemas Computacionais*; SBC: Porto Alegre, RS, Brasil, 2020; pp. 121–126.
34. 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). IEEE Computer Society, Cupertino, CA, USA, 22–25 August 2015; pp. 1–39.
35. Conti, F.; Rossi, D.; Pullini, A.; Loi, I.; Benini, L. Energy-efficient vision on the PULP platform for ultra-low power parallel computing. In Proceedings of the 2014 IEEE Workshop on Signal Processing Systems (SiPS), Belfast, UK, 20–22 October 2014; pp. 1–6.
36. Rossi, D.; Loi, I.; Conti, F.; Tagliavini, G.; Pullini, A.; Marongiu, A. Energy efficient parallel computing on the PULP platform with support for OpenMP. In Proceedings of the 2014 IEEE 28th Convention of Electrical & Electronics Engineers in Israel (IEEEI), Eilat, Israel, 3–5 December 2014; pp. 1–5.
37. Abate, F.; Sterpone, L.; Violante, M. A new mitigation approach for soft errors in embedded processors. *IEEE Trans. Nucl. Sci.* **2008**, *55*, 2063–2069. [CrossRef]
38. Herdt, V.; Große, D.; Le, H.M.; Drechsler, R. Extensible and configurable RISC-V based virtual prototype. In Proceedings of the 2018 Forum on Specification & Design Languages (FDL), Garching, Germany, 10–12 September 2018; pp. 5–16.

39. Barbirotta, M.; Mastrandrea, A.; Menichelli, F.; Vigli, F.; Blasi, L.; Cheikh, A.; Sordillo, S.; Di Gennaro, F.; Olivieri, M. Fault resilience analysis of a RISC-V microprocessor design through a dedicated UVM environment. In Proceedings of the 2020 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), Frascati, Italy, 19–21 October 2020; pp. 1–6.
40. George, N.J.; 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.