





Article

In-Pipeline Processor Protection against Soft Errors

Ján Mach ^{1,*} , Lukáš Kohútka ²  and Pavel Čičák ¹

¹ Institute of Computer Engineering and Applied Informatics, Slovak University of Technology in Bratislava, 812 43 Bratislava, Slovakia

² Institute of Informatics, Information Systems and Software Engineering, Slovak University of Technology in Bratislava, 812 43 Bratislava, Slovakia; lukas.kohutka@stuba.sk

* Correspondence: jan.mach@stuba.sk

Abstract: The shrinking of technology nodes allows higher performance, but susceptibility to soft errors increases. The protection has been implemented mainly by lockstep or hardened process techniques, which results in a lower frequency, a larger area, and higher power consumption. We propose a protection technique that only slightly affects the maximal frequency. The area and power consumption increase are comparable with dual lockstep architectures. A reaction to faults and the ability to recover from them is similar to triple modular redundancy architectures. The novelty lies in applying redundancy into the processor's pipeline and its separation into two sections. The protection provides fast detection of faults, simple recovery by a flush of the pipeline, and allows a large prediction unit to be unprotected. A proactive component automatically scrubs a register file to prevent fault accumulation. The whole protection scheme can be fully implemented at the register transfer level. We present the protection scheme implemented inside the RISC-V core with the RV32IMC instruction set. Simulations confirm that the protection can handle the injected faults. Synthesis shows that the protection lowers the maximum frequency by only about 3.9%. The area increased by 108% and power consumption by 119%.

Keywords: processor; soft error; functional safety; RISC-V; CPU



Citation: Mach, J.; Kohútka, L.; Čičák, P. In-Pipeline Processor Protection against Soft Errors. *J. Low Power Electron. Appl.* **2023**, *13*, 33. <https://doi.org/10.3390/jlpea13020033>

Received: 31 March 2023

Revised: 27 April 2023

Accepted: 8 May 2023

Published: 10 May 2023



Copyright: © 2023 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

Single event effects (SEEs), or soft errors, are random faults caused by radiation. Earth's atmosphere and magnetic field act as a protective shield against radiation, so the probability of soft errors in processors for terrestrial applications is significantly lower than in the processors used in space. The fault becomes a failure when it results in a loss of safety function or causes a violation of a safety goal. Functional safety generally recognizes two categories of faults: systematic and random. Systematic faults often arise from errors in development, manufacturing, or operation. Random faults are inherent to an application, use case, or operating environment [1]. A soft error is an example of a random fault. We will follow a taxonomy proposed in [2] to distinguish between fault, error, and failure. A processor failure means at least one external state deviates from the expected. This deviation is called an error, and the cause of the error is called a fault. Radiation can be defined as a set of particles, charged or not, that interact with the electronic system through an exchange of energy [3]. If this particle hits the material the circuit is built of, the energy is transferred into it, which could lead to a SEE or a cumulative effect. A single event transient (SET) is a temporary voltage spike at a node in an integrated circuit generated by a single particle ionizing the semiconductor and passing through or near a sensitive junction [4]. The amplitude and duration of the transient are determined by the ion species, its energy, and the fabrication technology of the microcircuit [5]. A storage element, such as a flip-flop, may capture this spike, so the unexpected value is saved. We refer to a single event upset (SEU) when the SET is generated within the storage element, leading to a bit-flip. Multiple bits upset (MBU) is an event when the hit of the particle causes more than

one SEU. This paper targets protection mainly against non-destructive soft errors, but the proposed scheme also provides some protection against permanent faults.

A fault is masked if it does not lead to an error. Various types of fault masking could happen inside the processor. Logical masking is when the fault could not logically cause the error. Any faults at other inputs are masked if an OR gate has one input connected to logical 1. We refer to electrical masking if the SET does not reach the storage element because it is attenuated. Temporal masking happens if the SET is not present at the input of the storage element when the input is captured.

Safety-critical applications, such as automotive and aerospace, require more and more computational power. To satisfy this requirement, processors are implemented in smaller fabrication technologies, allowing higher operational frequency, more transistors per chip, and higher performance. The problem is that they still need to have high dependability and robustness. An excellent study of how the susceptibility to SEE is changing across wide generations of fabrication technologies is presented in [6]. It also compares SOI vs. bulk substrates and planar vs. FinFET transistors. The primary outcomes are that the critical charge required to cause SEU decreases as technologies shrink. This also applies to a linear energy transfer from the particle. The saturated cross-section decreases, which is a good sign, but it seems like it has halted at the area of the ion track. The SET pulse monotonically decreases [6]. However, in advanced technologies, with the decrease of the propagation delay, the SET can more easily go through many logic gates. Thus, the latching probability increases [4]. When the circuit's frequency rises, the transient spike occupies a broader part of the clock period, leading to a higher probability of latching (lower probability of temporal masking).

This work proposes a novel protection scheme against soft errors, which will fully apply at the register transfer level (RTL) without requiring specialized hardened process technologies. The logic necessary to protect the core should have a minimum impact on the frequency, power consumption, and area. Large register files inside the cores are usually protected by error detection and correction codes (EDAC) but are susceptible to fault accumulation. Our proposed protection scheme will also include a proactive component, preventing fault accumulation in the register file without performance degradation.

The paper is organized as follows. Section 2 describes related work and the state of the art of radiation-protected processor architectures. In Section 3, requirements for the protection scheme are discussed. Section 4 describes the base unprotected RISC-V processor and integration of the proposed protection scheme. Verification, fault injection, and simulation results are described in Section 5. Results from synthesis, performance assessment, and final comparison of protection approaches are presented in Section 6. Section 7 concludes the article by discussing results and future works.

2. Related Work

The protection against soft errors could be realized at different levels of design, manufacture, or usage. Some processors are manufactured using dedicated radiation-hardened technologies, providing them with high resistance to radiation effects. We will refer to this type of protection as radiation hardening by process (RHBP). Such protected processors have limited use and high prices, and they lag several generations behind commercial off-the-shelf processors in terms of performance and power consumption [7]. These approaches generally concern modifications of doping profiles in devices and substrates, deposition process optimization for insulators, and the use of specific materials [4]. Radiation hardened by design (RHBD) processors are produced using commercial fabrication processes, allowing them to maximize their integration and performance capabilities. They offer similar or higher tolerance to radiation effects and lower costs than RHBP processors [7]. The hardening focuses on protecting the architecture itself, including several types of redundancy and EDAC. Some approaches leverage commercial process technologies with the protection applied at the technology library level by modifying circuits of gates or functional units.

The protection can also be realized at a system level. An example of this approach is a lockstep technique, where two or more processors running the same program are first synchronized to start from the same state and receive the same inputs. Therefore, the states of the processors should be equal every clock cycle unless an abnormal condition occurs. The retrieval and comparison of processor states are performed after the program has been executed for a predefined amount of time or whenever a milestone is reached during program execution (e.g., data should be written to memory). When states differ, the execution of the application must be interrupted, and processors must be restarted from the previous error-free state [8]. If the lockstep architecture is selected, it is also possible to delay the execution of one core to target temporal diversity. For example, it is possible that lock-stepped cores can operate 1.5 or 2 cycles out of phase to mitigate common mode failure in clocking [1]. Authors in [9] presented a dual-core lockstep technique where the checker core can run freely behind the main core within the constrained boundaries of clock cycles.

Fault tolerance can also be achieved through software techniques. These techniques do not rely on hardware redundancy or implementing additional modules that can detect and correct errors. The fault tolerance is achieved by modifying the software running on the processor. The cost of these techniques is typically low. However, this approach usually decreases the system performance since the processor must execute longer programs with more instructions than in the original code [10].

Section 2.1 depicts some of the state-of-the-art processors currently in use but also solutions in the advanced development stage, and information about them is available. This section also provides a comparison of these processors. Section 2.2 highlights some proposals to protect processors or approaches to protect some parts of processor architecture against soft errors. Some pros and cons of individual approaches are discussed in Section 2.3.

2.1. State of the Art

This section describes the soft-error protection of selected state-of-the-art processors. Their comparison is shown in Table 1. An example of an RHBP-protected processor is a second-generation RAD750 [11], which features PowerPC architecture. It is an evolution of the first generation, protected by RHBD techniques at 250 nm nodes.

Some of the most well-known RHBD processors are based on the LEON processor architecture with the SPARC-V8 open instruction set. An example of such a processor is an AT697F. It is manufactured using a 180 nm CMOS process and reaches a maximum frequency of 100 MHz. Each on-chip register is implemented using triple modular redundancy (TMR), and an independent clock tree is leveraged for each of the three registers, making up one TMR module. This approach helps to protect against SET in the clock tree. A 7-bit EDAC checksum protects each record in the register file. The checksum is validated every time a fetched register value is used in an instruction. When detecting a correctable error, the flawed data is corrected before being utilized, and the corrected value is also stored back in the register file. However, the logic between pipeline registers is not redundant. A skew can be set at each clock tree to protect the logic against SET. The disadvantage of this approach is that it reduces the maximum operating frequency to 83.3 MHz [12]. Since the processor is implemented in the larger fabrication technology and runs at a lower frequency, it is less likely that the SET will affect the outputs of the inter-stage logic.

A quad-core GR740, running at 250 MHz system frequency, is one of the most recent space-grade processors. It also features the SPARC-V8 instruction set but is implemented in the 65 nm CMOS technology platform for space application C65SPACE, which includes hardened and non-hardened flip-flops. Register files of each core are implemented by non-hardened flip-flops protected by a Block TMR correction scheme, with bit-by-bit voting on register read data outputs to mask SEUs. To avoid sensitivity to potential uncorrectable MBUs, individual modules of the Block TMR have adequate spacing among them. Peripheral's SRAMs are protected by TMR, caches with parity and EDAC codes.

Sequential logic and all other memory cells in the GR740 are composed of hardened flip-flops that do not require mitigation at the design level. This is achieved by the RHBD approach at the ASIC library level [13].

The ARM Triple Core Lock-Step (TCLS) [14,15] is an example of architecture that includes three Cortex-R5 cores running in lockstep. A TCLS Assist Unit coordinates the cores and acts as an error propagation boundary in the system. It contains majority voter, error detection logic, resynchronization logic, and control registers. All outputs of the cores are compared by the TCLS Assist Unit each clock cycle. Upon detection of a mismatch in a single core, the TCLS Assist Unit can trigger error recovery actions, which consist of resynchronizing three cores again into an architecturally safe state. This procedure takes more than 2000 clock cycles. Each core in this architecture has its separate clock tree but shares the interrupt controller, AMBA network interconnect unit, and the TCM and instruction/data caches. Authors in [15] used the 65 nm C65SPACE technology library for ASIC implementation, which includes hardened and non-hardened elements. The hardened elements were used for the TCLS Assist Unit and non-hardened in the other SoC components, including the Cortex-R5 CPUs, peripherals, and TCMs. The resulting frequency of the system was 311 MHz.

ARM has implemented the lockstep approach into their 4-core Cortex A76AE processor, which allows operation mode selection. In a Split-mode, each core executes independently. In a Lock-mode, two pairs of cores are created. One of the cores in the pair serves as a redundant copy of the primary function core [16]. It is important to note that each core has its cache, which brings enormous area and power consumption overhead in lock mode because we could leverage only two cores for execution (two cores are redundant, so they do not increase performance).

Table 1. Comparison of processors.

Processor	Instruction Set	Protection Approaches ¹	Node Size [nm]	Frequency [MHz]
AT697F [12,17]	SPARC-V8	TMR, RHBD, clock skew	180	100
GR740 [13,17,18]	SPARC-V8	RHBD, library	65	250
TCLS [14,15]	Armv7-R	TCLS, library	65	311
SAMRH71 [19]	Armv7E-M	RHBP	150	100
RAD750 1st gen. [11]	PowerPC	RHBD	250	133
RAD750 2nd gen. [11]	PowerPC	RHBP	180	200
NOEL-V [20]	RISC-V	RHBD, library	28	800

¹ All processors/systems leverage EDAC to protect parts of the architecture/memories.

2.2. Concepts and Proposals

The protection of the register file can be very power and area-consuming, particularly for the SPARC architecture. To ease this problem, authors in [21] presented the robust-register-caching approach and demonstrated it at 32-bit LEON2 processor architecture, which uses 136 registers in the register file. The most vulnerable registers are cached in small circuit-level protected memory elements instead of directly saved into the register file, which is protected with parity bits only. The motivation behind this approach is to reduce the power consumption and area overhead, which come from protecting the whole register file by single-error correction, double-error detection codes (SECDED).

An architecture that leveraged the lockstep approach of two cores with all its consequences is described in [22]. The two cores execute the same application, which must be divided into blocks separated by verification points (VP). When the execution reaches a VP, processor status is written into the memory, and the execution is locked. In sequence, an additional checker IP generates an interruption to access the processor's registers. Then, the checker compares the outputs and the register files of both CPUs. If no difference is detected, the system is considered to be in a safe state, and a new interruption is launched for each CPU to perform a checkpoint operation and save the processor's context. If there

is any mismatch in the results, an interruption is generated to recover the processors using the rollback mechanism. Ultimately, the checker writes a flag to both memories, unlocking the processors. This cycle executes in a loop until the end of the application.

A dynamic lockstep architecture is presented in [23]. In this technique, unrelated homogeneous cores can independently accept a request to join the lockstep to process a critical task. Once this task is executed, processors release themselves and are available for other tasks. The NIOS II cores were used to present this architecture. As the authors reported, this architecture has some caveats. The leveraged cores utilize branch prediction based on execution history. This brings individual cores into the lockstep mode with different internal states, which can result in unsynchronized execution. A similar idea led the authors in [24] to introduce dynamically coupled cores, where the operating system could select two cores to execute the assigned thread redundantly.

The lockstep between two threads in a multithreaded core was presented in [25]. The authors implemented the fine-grained multithreading into two pipelines. In the first cycle, pipeline A executes the master copy of program 1, and pipeline B executes the master copy of program 2. In the next cycle, pipeline A runs the slave copy of program 2, and pipeline B executes the slave copy of program 1. The master and slave copies of the program are executed in the following cycles, so a compare mechanism is implemented for fault detection. In some cases, it can return to the correct state, so the execution of the program does not need to be restarted. However, when the control states or program counters in the execution of master and slave copy mismatch, the execution of the affected program is restarted. The protection by running redundant threads at a multithreaded processor has also been presented in [26]. The authors leveraged three threads to implement a so-called buffered TMR. Then the architecture improved performance in [27], where the third thread was started only on-demand if the first two threads produced different results.

The RTL level fault-tolerant microprocessor architecture (SHAKTI-F) has been proposed in [28] and presented in a RISC-V processor with a RV32I instruction set. The solution highly leverages SECDED codes. They protect instruction and data memories as well as the program counter and register file. They also protect registers separating individual pipeline stages. The detection and correction logic is placed before the combinatorial logic of each stage, and the parity bits are generated at the output from the combinatorial logic before writing to the following registers. The authors highly focused on protecting functional units in the execution stage. Each functional unit is duplicated, and the architecture can trigger re-computation in case of discrepancy. In case of permanent fault, localizing and isolating the faulty functional unit and continuing with reduced fault tolerance is possible.

2.3. Advantages and Disadvantages

Nowadays, approaches to protect the CPU system from faulty behavior are focused on protecting commonly used and verified cores in non-harsh environments or intended for non-critical applications. They keep the architecture of the base core the same and change very little. Proving that the system is fail-safe is a costly and time-consuming process. So, it is an advantage that producers can only focus on verification and testing of the protection of the whole system and do not need to design and verify their core. On the other hand, the downside is that such a system comes with a deterioration of power, performance, and area (PPA) metrics. It could be more effective if the protection were integrated directly into the architecture.

System approaches, such as lockstep with two replicas of the base core, cannot localize fault, and the affected process needs to be rolled back to a known safe state. This degrades performance as the application needs to create backups periodically. It also comes with additional memory requirements to save the backup. The performance can be maintained in the case of a triple-lockstep system such as TCLS, but such architectures come with huge area and power consumption. Lockstep architectures also come with another significant downside. As replicas of base cores are used, large components, such as predictors not critical in terms of correct execution, are replicated too. Register files and caches are often

replicated, although a single replica with ECC protection would be more effective. As discussed in Section 2, some implementations leverage radiation-hardened libraries to protect the core. These include either protecting circuits of individual gates or selecting specialized materials and processes to protect an architecturally not protected core. This approach comes with a high cost for the final product, which often lacks several generations in terms of performance behind cores based on commercial process libraries.

3. Requirements for the Protection Scheme

3.1. Predefined Properties

This work aims to inherently include the protection into the architecture of the core at the RTL level so that it can leverage the full potential of commercial off-the-shelf process libraries. Any additional logic implemented into the pipeline of the core should have a minimal impact on the maximal frequency, power, and area consumption. Huge parts of the architecture, not directly responsible for correct execution, such as a branch predictor, will not be protected at all. The response to a fault and subsequent recovery should be fast, and the application will not need to create backups.

3.2. Fault Detection in the Pipeline

The primary purpose of the protection scheme is to prevent a fault from becoming an error. This means that the protection implemented into a system should prevent the fault generated inside the system from causing unintended values at the interface of the system. By intended value, we mean a value not affected by the fault or a value that cannot cause any functional change in the processor's surroundings. In terms of the processor, the interface can be a connection to the bus. For example, suppose an execution of the store instruction is affected by a fault, and the protection scheme does not allow initiating a transfer to the bus. In that case, the fault does not cause any functional change because no transfer is initiated. As soon as the fault is detected, the protection scheme must re-execute the store instruction to send the correct transfer to the bus. In this example, the fault causes only a delay in execution.

Real-time systems are strict in timing, so fault detection and recovery must be as quick as possible. Restarting the instruction execution in short pipelines and with already cached instructions can take only several clock cycles. This fast recovery should be fine with timing, especially if the core runs at gigahertz frequencies. It may seem that each instruction, which execution is affected by a fault, must be restarted. Let us consider another example. The store instruction initiates a request to the bus, and the fault causes some bit flip in the pipeline's control registers as we await confirmation of the successful transfer. We assume the request was sent correctly since the instruction was not restarted before the request. The problem is that we have detected some anomalies in the control registers, but we cannot restart the instruction because it would cause another request to the bus, which would be considered an unintended value at the interface. The protection scheme brings up some constraints, which can be summarized as follows:

1. A fault should be detected before the data bus transfer is initiated
2. Whenever the fault is detected, the instruction execution should be restarted
3. When the transfer is initiated, we must ensure that the instruction will be finished correctly and the fault masked

Those three basic requirements separate the processor pipeline into two sections, shown in Figure 1. The first is before the place where the transfers to the data bus are requested. It is enough to detect the fault in this section. The remaining section lies from the point the requests are made up to the last point of execution of the instruction. The protection scheme must ensure that the instruction is finished correctly and mask any fault in this section.

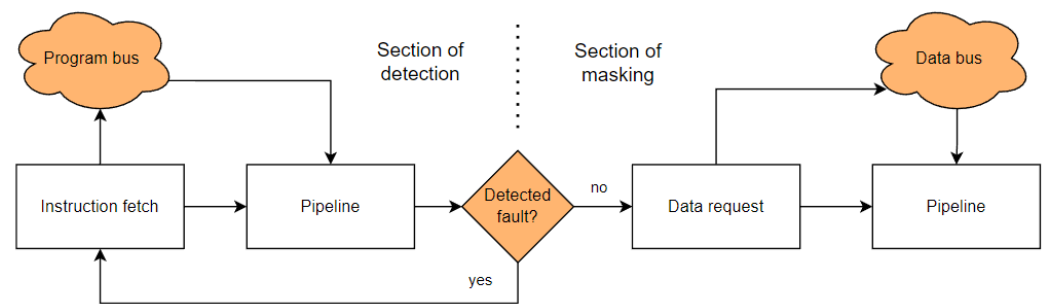


Figure 1. Separation in the protection scheme.

3.3. Protection of In-Core Memories

Processors also contain memory elements, such as register files, state machines, control and status registers, and other individual registers, which are not directly responsible for the execution of the instruction but contribute to it and reflect the current state of the processor. Some memories are necessary to execute instructions, e.g., the register file. Their protection is often provided by error-correction codes because replication would cause significant area and power penalties. The fault is detected and usually corrected if a faulty record is read. This means the fault does not have to manifest immediately but only when an instruction uses the faulty register. If individual records are not periodically read out, the memory (or register file) could be subject to fault accumulation, resulting in an uncorrectable or even undetectable error. The register file in our architecture will be protected by an error-correction code but will also include a proactive scrubbing mechanism to prevent fault accumulation.

Faults in other in-core memory elements, such as tables inside the branch predictor, may only affect the performance. These tables are often larger than register files, so the fault probability is higher. Non-protected cores can handle mispredictions and correct them. If a fault arises in the predictor, it will corrupt the saved information about previously executed branches, affecting the success of the upcoming predictions. Any fault inside the predictor can affect only performance in terms of more pipeline flushes. This means we can afford to leave the whole predictor unprotected because it will not impact the correctness of execution. Control and status registers are essential memory elements that hold the currently running application's settings and privilege information. They are less likely to be read or written, and we must ensure the fault will never cause a loss of their content. To ensure this requirement, TMR will protect all these registers.

4. Proposed Solution

We have developed a 32-bit processor, code named Hardisc, to validate the proposed protection scheme and present its potential. It is based on the free and open instruction set architecture RISC-V. One of the main advantages of this instruction set is its modularity so that it can be used in various applications. Apart from the base 32-bit integer instruction set (I), the Hardisc also implements extensions of compressed instructions (C) and multiply and divide (M) instructions. An actual designator of the implemented instruction set is RV32IMC. To support privilege modes and standard settings of the core through control and status registers (CSR), the Hardisc also supports the Zicsr instructions. Only the machine mode is currently supported. The C extension is essential since 50–60% of the RISC-V instructions in a program can typically be replaced with these 16-bit instructions, resulting in a 25–30% code-size reduction [29]. The M extension is necessary for higher performance and the smaller size of the code. These extensions have nuances that affect the final architecture. Recently proposed hardened RISC-V architectures [28,30] describe the protection only on the processors with the base integer instruction set. The Hardisc is described in a SystemVerilog and maintained in an unprotected and protected version. Both versions are functionally identical, but the protected one leverages the proposed hardening scheme. The unprotected one serves for the comparison of the impact of protection on

the power consumption, performance, and area of the processor. The protection logic is separable so that it may be enabled or disabled before synthesis.

4.1. Unprotected Pipeline

The Hardisc has a six-stage pipeline and Harvard architecture. It uses the AMBA 3 AHB-Lite bus protocol [31]. From the beginning of the development, the pipeline was designed to allow high-frequency operation. To shorten register-to-output paths, addresses of requests for data and program buses are computed in a cycle (or a stage) before the requests are made. It is necessary to mention that the architecture of the base unprotected pipeline has been developed to be protected lately, so some design approaches were selected with this bias. An overview of the unprotected pipeline of Hardisc is shown in Figure 2.

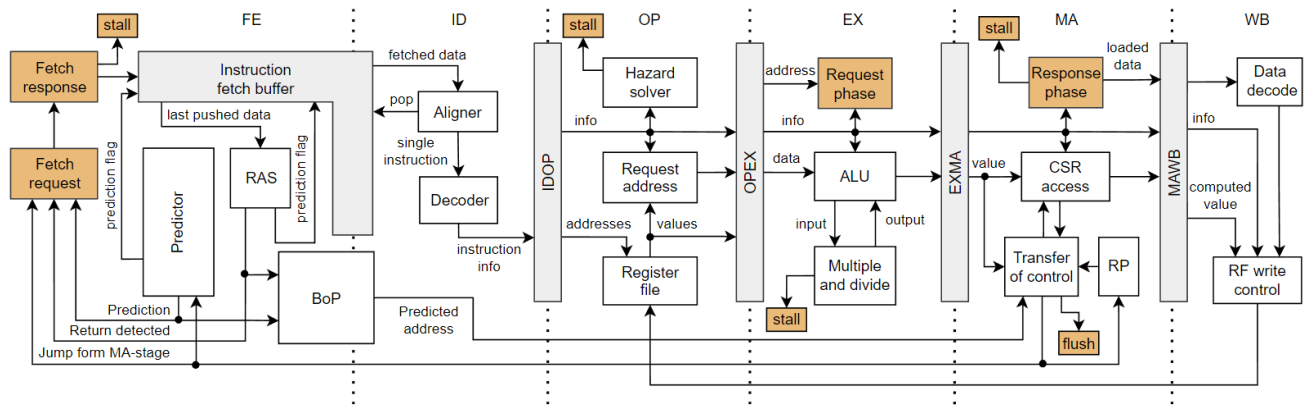


Figure 2. Overview of unprotected pipeline.

The first stage (FE) fetches instructions from a program bus into an instruction-fetch buffer (IFB). This stage also contains a predictor, a return address stack (RAS), and a specialized buffer of predictions (BoP). Section 4.1.2 describes the whole prediction scheme. The FE stage can be separated into two stages: one cycle is used for the address phase of the request, and the second for the data phase. For simplicity, we will mark the whole logic as the FE stage. In the second stage (ID), instruction decoding is performed. Since the Hardisc also supports compressed (16-bit) instructions, the instruction must be aligned before decoding. The alignment is conducted inside the Aligner, which can pop one record out of IFB and output exactly one instruction (32- or 16-bit) for the Decoder. If the popped record contains parts of two instructions, the remaining part is saved inside the Aligner. The third stage (OP) prepares operands for the next stage. It reads the register file and computes an address for access to the data bus. This stage also checks data hazards, which are not solvable by forwarding, so it inserts bubbles if necessary. The intended operation of the instruction is executed in the fourth stage (EX). It contains an ALU with sequential logic to execute multiplication and division instructions. The multiplication has a variable-long computation time and lasts between 2 and 18 clock cycles. The division algorithm still needs to be optimized; it always takes 34 clock cycles to compute the result. A stall of the pipeline is signalized if necessary. The EX-stage is also leveraged to perform the address phase of the data bus request. The fifth stage (MA) is responsible for finishing the bus requests, and it signalizes a pipeline stall if the access lasts longer. This stage manages all control transfers to other addresses and accesses CSR registers. The sixth stage (WB) is responsible for decoding the data returned from the data bus and writing the instructions' results into the register file.

4.1.1. Reset Point

Knowing the address from which the new instruction should be fetched is necessary. However, most instructions do not require knowing its address to perform the intended operation. The main exceptions are control-transfer instructions, such as branches and

jumps. Holding the address in each stage and row of IFB is a waste of area and power. These downsides would be even more magnified in the protected architecture, and we will discuss them in the following sections. To prevent these problems, we leverage a mechanism in which the main element is a 32-bit register called a reset point (RP), placed into the MA-stage. From a programmer's perspective, it acts as a program counter (PC). The RP is loaded with a boot address during reset. Each finished instruction stores an address of the following instruction in this register. This also implies that whenever the valid instruction is in the MA-stage, the RP contains its address. So, each instruction, which needs to know its address, uses the value of RP. Control-transfer instructions write a target address into the RP, and the computational instructions increment the RP by 2 or 4, depending on their width (16- or 32-bit). This register also provides an address of return when an interrupt is taken, and the jump to the trap handler must be performed. So, interrupts can be served immediately, and waiting for valid instruction in the MA-stage is not required.

4.1.2. Predictions

One of the largest components inside high-performance processors is a predictor, which predicts the outcome of the control-transfer instructions so the pipeline does not need to be flushed. The current implementation of the predictor in the Hardisc leverages a simple prediction scheme based on a history of predictions. It consists of two parts. The first part predicts the outcome of conditional branches. It includes a branch target buffer to save executed branches and a branch history table with saturation counters to record the history of prediction. The second part predicts the outcomes of unconditional jumps and contains a jump target buffer. Only a small part of the instruction address is saved in the individual rows of target buffers. The predictors are usually more complex and contain many register files with hundreds or thousands of bits. The predictor may predict a jump from an instruction, which does not perform the transfer of control. Thus, it is necessary to have a mechanism to detect and correct these mispredictions. Each instruction from which the prediction is performed gets a prediction flag. If such instruction gets into the MA-stage and the prediction is not correct or allowed for this instruction, a jump to the address saved in RP is executed. The faulty record in the predictor is erased, so it does not predict transfer of control again.

To check predictions, it is necessary to hold the predicted target address and the information that the prediction was performed (the prediction flag) until the decision about the correctness of the prediction in the MA-stage is made. It is improbable that the prediction would be made from each instruction present in the pipeline simultaneously. Therefore, it is inefficient to propagate the prediction information through the pipeline and have dedicated registers for this information in each stage and row of IFB. Instead, the Hardisc contains a separate FIFO-based buffer of predictions (BoP), which holds the predicted target addresses. The prediction flag propagates with the instruction through the pipeline. The target address is pushed into the BoP whenever the prediction is performed. On the other hand, the prediction is performed only if the BoP has free space. If the instruction from which the prediction was performed propagates into the MA-stage, the target address is popped from the BoP and evaluated. Each flush of the pipeline also flushes the BoP. In our testing, it was enough to have two rows inside the BoP without noticeable performance loss. Since three pipeline stages and a four-row IFB separate the path between a prediction and evaluation, it dramatically reduces the count of registers.

The RAS is also a predictor, typically in the form of a circular buffer with only several entries, so it occupies less area. The predictions are not based on the instruction address but on the data fetched from program memory (the latest saved into the IFB). If the RAS detects a return instruction, it predicts a return address. The used IFB has a special output port for the latest pushed data, which the RAS leverage to evaluate the instruction. The Hardisc supports an extension for compressed (16-bit) instructions, so the RAS must count with the alignment of instructions. The IFB also contains a special input port leveraged

by RAS to override the information about predictions from the latest pushed instruction. The return address is pushed into the BoP in the same way as the target address from the predictor. The RAS has a higher priority.

4.2. Application of Protection Scheme

We must meet the three basic requirements described in Section 3.2 to successfully implement the protection scheme. The address phase of requests to the data bus is executed in the EX-stage, so this stage is where the pipeline is separated into two sections. In the first section (stages before the EX-stage), it is enough to check whether a fault influenced the execution of the instruction. When the request is executed in the EX-stage, the instruction needs to be finished, so in the next section (stages following the EX-stage), we need to ensure the instruction is completed faultlessly. To fulfill these requirements, we replicate the pipeline inside both sections. This can be seen in a simplified form in Figure 3. The first section is only duplicated (pipeline d0 and d1), as it is enough to detect a fault. The second section is triplicated (pipeline t0, t1, and t2) because we need to be able to mask out the fault. Register outputs from pipelines d0 and d1 are compared in the EX-stage. Upon detected discrepancy, the instruction is marked as corrupted, and possible transfer to the data bus is not initiated. The control logic inside the EX-stage is triplicated. Each replica gets output from both comparators of discrepancy. This ensures that the fault will be detected if one of the comparators fails due to SET. The bus request information is taken from pipeline d0, but the transfer is not initiated if the discrepancy exists. If at least one of the comparators signalizes the discrepancy, the control logic in each replica marks the instruction as corrupted. If such an instruction propagates into the MA-stage, it is restarted by a jump to the address saved in the RP.

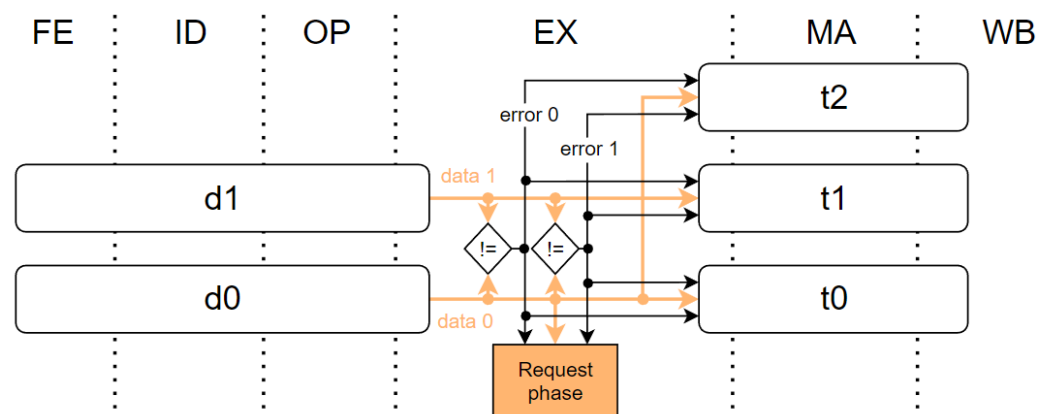


Figure 3. Separation and interconnection of the protected pipeline; FE, ID, OP, EX, MA, and WB are names of individual pipeline stages; d0 and d1 are replicas of duplicated parts of the pipeline; t0, t1, and t2 are replicas of triplicated parts of the pipeline.

The EX-stage expands duplicated logic into triplicated. The pipeline d0 is connected to t0 and t2; d1 is only connected to the t1 pipeline. The ALU-related instructions do not access the data bus, so we do not need to mask out faults in it. This is an important property as we do not need to triplicate the whole ALU, which occupies a large area and has high power consumption. The EX-stage contains two replicas, ALU0 and ALU1, because we need to detect faults. The ALU0 receives inputs from the d0 and the ALU1 from the d1. The output of the ALU0 is connected to the pipelines t0 and t2, and the output of ALU1 is connected to the t1. Such an interconnection has less routing and better timing results. If the output of ALU0 was connected to each pipeline and the ALU1 served only for comparison, we would need an additional comparator at the outputs of the ALUs (in the EX-stage) before saving to the following registers. Instead, the logic inside the MA-stage compares product registers of pipeline t0 and t1, written from the EX-stage, to detect a fault in ALUs. The instruction is marked as corrupted and directly restarted if different results are stored in these registers.

All registers between the EX-stage and MA-stage and between the MA-stage and WB-stage are protected by TMR. Each copy of a register in the TMR circuit is driven by one of the tX pipelines. The voter at the output of the TMR is also triplicated and drives one tX pipeline. Any fault in the MA-stage logic is masked by majority voting in the WB-stage. The MA-stage also contains the RP register and CSR registers, which are protected by TMR too. The response from the data bus is pushed into each pipeline replica in the MA-stage.

The pipeline d0 determines a request to the program bus, and the fetched instruction is saved into IFBs of d0 and d1 separately. The fetch stage preserves the request address until the bus responds. So, before the data are written into the IFBs, addresses of both pipelines are compared, and a flag is saved with the incoming data to signalize discrepancy. If d0 and d1 are different at the time of the request, the discrepancy is detected in the following stages. The protected core contains only one instance of the predictor, which gets the currently fetching address only from pipeline d0, but its prediction is signaled to both pipelines. The address in pipeline d1 should be the same. If it is affected by a fault and contains a different address, the pipeline will be restarted, and the prediction will be discarded. Since our unprotected pipeline can detect all types of misprediction, we can afford to leave the whole predictor unprotected. The same situation occurs for RAS. It is fed from d0, and only one instance is present. Any fault, which occurs inside the predictor or RAS, can lead only to a single misprediction. Information that the prediction was performed (the prediction flag) must be protected. So, the flag is saved to the IFB of both pipelines. The target address is saved into the BoP. Any fault inside the BoP corrupts the saved predicted address, so the misprediction will be detected during evaluation and the instruction will be restarted. This means the BoP does not have to be protected at all. So, we have only one instance of BoP in the architecture. The fetch stage and prediction in the protected pipeline are shown in Figure 4.

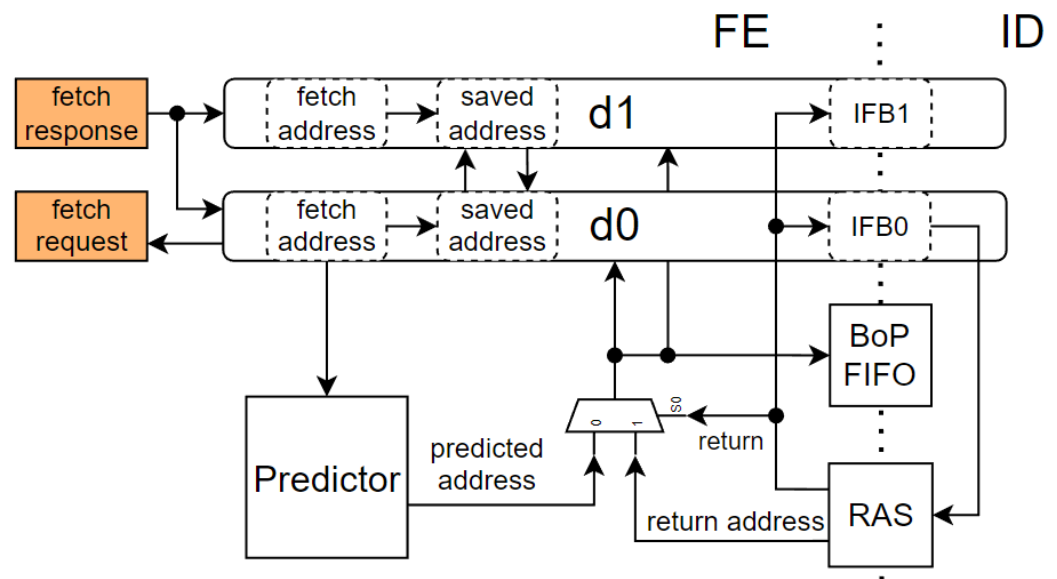


Figure 4. Fetch stage and prediction; FE and ID are names of pipeline stages; d0 and d1 are replicas of duplicated parts of the pipeline; IFBx is Instruction Fetch Buffer in x-th pipeline replica; BoP FIFO is Buffer of Predictions, which behaves like First-In-First-Out buffer.

Since we replicate pipelines, avoiding the introduction of a common failure mode is essential. This means we must also separate the clock and reset trees. Single clock and reset trees are insufficient since fault within them would result in errors in each redundant pipeline or TMR register. The Hardisc has three dedicated ports for clock and reset signals. Clock0 is for pipeline d0 and t0, clock1 for d1 and t1, and clock2 for t2. Clock2 is utilized for clocking the least number of registers and units in replicated pipelines, so it is also responsible for clocking the register file. The predictor and RAS are clocked by clock0. The

reset ports are utilized similarly. To support more control over the protection scheme and some specific functions of the Hardisc core, a new custom CSR register named `hrdctrl` was implemented. Its purpose will be described in the following sections.

4.3. Protection of Register File

A SECDED code is leveraged to protect the register file. The syndrome is checked during a reading of the register file in the OP-stage. The checksum is generated in the WB-stage inside each tX pipeline separately. As the register file contains only a single write port, a TMR logic is used to select valid data for the write port from the three pipelines. The instruction is marked as corrupted upon detecting a correctable error (CE) in the requested register during reading. Its execution is restarted by a jump to the address saved in the RP from the MA-stage. The read value is routed to the OP-stage directly from the register file in parallel with syndrome evaluation and correction logic. This means that a critical path due to correction logic is not present.

A configurable automatic correction mechanism (ACM) is implemented, which helps decrease the probability of fault accumulation. Integration of the ACM into the pipeline is shown in Figure 5. The ACM makes sure the faulty register is corrected before the instruction propagates again back to the OP-stage. It reports information about the correctness of the read data to the OP-stage logic. The instruction is marked as not correctable upon detection of an uncorrectable error (UCE). In this case, an exception is reported if the instruction propagates to the MA-stage. The register file has two read ports, with addresses in pipeline registers (RS1 and RS2), written from the ID-stage. The ACM may be configured by the `hrdctrl` CSR register to automatically leverage the unused read ports to search and correct the faulty records in the register file. If the instruction in the ID-stage does not require a particular read port, or the OP-stage is empty, the ACM may change the read port address coming from the decoder. The new addresses are determined by a value of 5-bit registers (RS1x and RS2x), which are incremented in a round-robin fashion whenever the ACM changes one of the read addresses. This helps to periodically check all registers, thus lowering the faults accumulation. For simplicity, Figure 5 shows RS1 and RS2 registers only from pipeline d0, but these addresses are also changed in the d1 pipeline. Addresses for read ports are taken from the d0 pipeline, and read values are pushed to both the d0 and d1 pipelines. Each time a CE is detected, the ACM saves the corrected value to the internal registers to be written back to the register file in the following clock cycles. The ACM will wait with the write until instructions from the WB-stage do not require the single write port. A logic inside the ACM is duplicated to protect it from faulty behavior. If the copies of the ACM logic differ, the correction process is terminated, so the faulty value needs to be detected again to be corrected.

Programs/compilers for RISC-V architecture should use registers according to the calling convention described in [32]. This convention assigns to each register in the register file some attributes. Some registers are intended for temporary values, so they are not expected to hold the value after function calls. Some registers hold the value across calls of the functions. One register is dedicated to preserving a return address from a function, and one holds the address of the boundary of the stack. This convention leads to different registers usage, meaning some registers are used more frequently than others. Some sections in a program may leverage only a few registers. If such a section is a loop, unused registers can be untouched for several thousand clock cycles. If such a register contains a UCE and the ACM detects it, an exception is not violated if some instruction does not require the value of the faulty register. This can lead to a potential issue. The UCE could become undetectable if another fault arises within this register when the register is not required for a long time. This could then lead to incorrect program behavior. The ACM can also be configured as an interrupt source to prevent these situations. When these interrupts are enabled through custom `hrdctrl` CSR, the interrupt is raised whenever some UCE is detected. This provides an additional layer of protection. If necessary, the software can save the content of each register to memory (create a backup) periodically or before some

function call. Then the backup can be used to restore the state of the processor as soon as the UCE is detected.

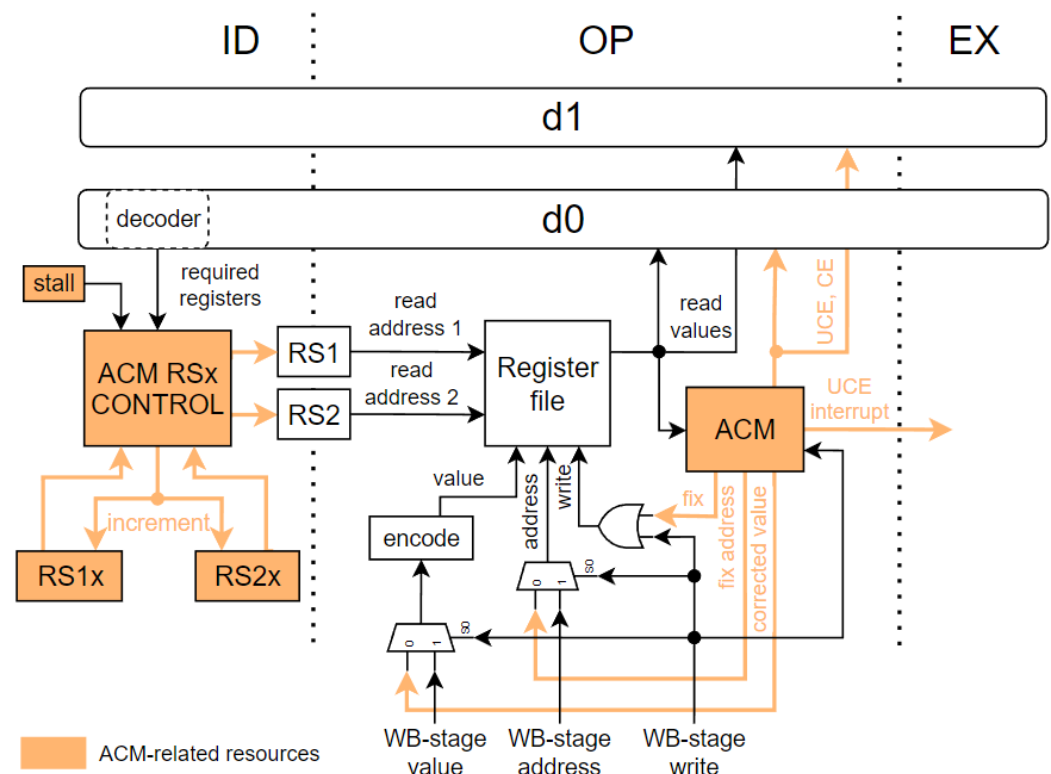


Figure 5. Integration of ACM; ID, OP, EX, and WB are names of pipeline stages; d0 and d1 are replicas of duplicated parts of the pipeline; ACM refers to Automatic Correction Mechanism; RS1 and RS2 are address registers for the register file; RS1x and RS2x are auxiliary registers of ACM.

We have analyzed how often instructions in one iteration of a CoreMark benchmark leverage read ports of the register file. We found that each two executed instructions require approximately three values from the register file. This means that at least every second clock cycle, the ACM can leverage one read port to check one additional register. The ACM also leverages read ports whenever the OP-stage is empty due to flush or memory latency. It can also alternate the addresses in registers RS1 and RS2 whenever the OP-stage is stalled, and the instruction in this stage does not require both read ports. Let us say the EX-stage executes some time-consuming operation (e.g., division), and the OP-stage contains an ADDI instruction, adding immediate value to one register. The first read port is used to read the required source register, but the ACM can leverage the second port to check other registers. The ACM can check eight registers if the pipeline is stalled for eight clock cycles. As it is evident, this approach could lead to increased dynamic power consumption. To solve this issue, we included the possibility of disabling proactive change of RS1 and RS2 registers by custom hrdctrl CSR. The ACM will still search for fault, but the address register will not be changed if not needed by the instruction. So, the application can distinguish what level of protection the program needs.

4.4. Protection of External Memories

External memories or caches are often protected with error correction codes. This means that whenever the data are written to the memory, a checksum is generated by the core and written to the memory. The memory provides checksum and data whenever the data are requested, so their correctness can be validated. We have yet to implement this into the Hardisc processor, but we propose how and where the checksum generation and correction/detection logic should be implemented. The program bus is used only to read data from memory, so only a detection/correction logic is required. This logic should be

placed inside the aligner, so it would be possible to precisely distinguish which instruction is corrupted. The protected architecture contains an aligner in both the d0 and d1 pipelines. An exception with a unique code should be reported as the instruction enters the MA-stage so that the software can select the following action.

The data bus is also used to write data to memory, so the checksum generation logic is required. It should be placed inside the EX-stage and triplicated, so one replica will be inside each pipeline (t0, t1, and t2). The error detection/correction logic of the read data should be placed into the WB-stage to ensure it will not constrain the maximum frequency due to the long input-to-register path. The read data should be corrected before writing to the register file if it contains a correctable error. If data are not correctable, we propose to generate a non-maskable interrupt. The problem of fault accumulation, which could lead to uncorrectable or even undetectable errors, is also present in the memory blocks. External memory scrubbers are used to lower such probability. The protection of external memories is outside the scope of this work, but the known approaches should be easily implementable as a complement to the proposed architecture.

4.5. Protection against Permanent Faults

The Hardisc is protected against permanent faults too. These faults manifest as repeated restarts of instructions, and the Hardisc can detect them. The maximum number of instruction restarts can be configured in the custom hrdctrl CSR. If this function is enabled, the Hardisc can set the dedicated output port p_unrerr to 1 to signalize unrecoverable errors. The hrdctrl register can also be configured to disable all predictors first, and if the unexpected behavior persists, then the output port is set. A decision diagram of control of the p_unrerr port is shown in Figure 6. This output port can be checked by an external component, which can try, for example, to reset the power supply.

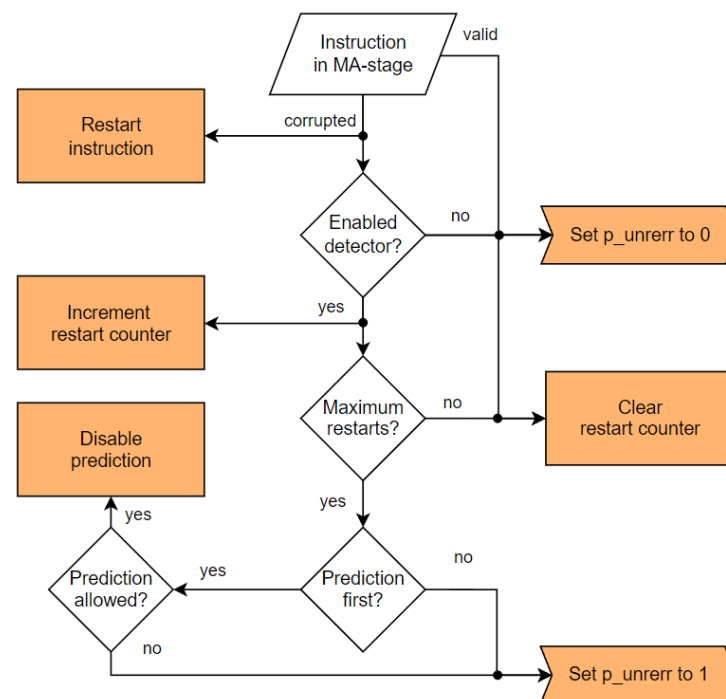


Figure 6. Control of the p_unrerr port.

5. Testing and Verification

Hardisc is described in a SystemVerilog RTL language and maintained in a protected and unprotected version. The protection and other parameters, such as IFB and RAS depth or predictor size, are configurable. All RTL simulations were performed in ModelSim.

5.1. RISC-V Compliance

Since we have developed our own RISC-V core, it was necessary to check compliance with the RISC-V specification. To do that, we utilized a random instruction generator RISC-V-DV [33]. It allows for generating groups of tests with different focus and target instruction subsets (in our case RV32IMC). During each test run, a trace log was generated and compared with the trace generated by running the same test on the Spike RISC-V ISA Simulator [34]. One hundred tests from 8 different groups were selected, which resulted in 95.8% functional coverage of the target instruction set. A trace generated by the Spike and Hardisc (protected and unprotected) was the same for each test. This means that our core executed instructions as expected. It does not mean the core is fully verified, but it shows up that the novel architecture and protection presented in this paper can be reasonably and legitimately applied to the RISC-V processor.

5.2. Fault Injection

The RTL description of Hardisc supports the insertion of faults into all registers of the core to simulate SEU. We can also select wires at which the SET is generated, including all clocks and reset trees. The faults are generated randomly in each bit of register or at the wire. By fault insertion, we mean bit-flip of the register or bit-change of the wire. A fault generation is evaluated for each bit every clock cycle. The probability of fault generation is configurable by option. Some registers and wires are grouped; we can choose groups with fault generation enabled. This allows us to test individual protection mechanisms. Available fault-insertion groups are register files, predictor/RAS, control and status registers, clock/reset trees, and output of ALUs.

5.3. Simulation Results

To check how the protection mechanism performs during the execution of the program, we chose EEMBC's benchmark CoreMark [35]. Table 2 shows the parameters of Hardisc used for simulations. Since the predictor and RAS are not protected and feed both the d0 and d1 pipelines, it was necessary first to deeply test the logic, which detects and handles predictions. Depending on the configuration, the predictor and RAS contain from several hundred up to thousands of bits in registers. The unprotected version of the Hardisc should be able to handle all misprediction. This means that if faults are enabled only inside the group of predictor/RAS, the Hardisc should successfully finish the program. Generated faults should affect only performance since more predictions will be incorrect. The predictor and RAS contain 769 flip-flops in total. With the selected fault-generation probability, a single fault was generated every 100 clock cycles on average during the whole execution of the CoreMark program. The program has finished correctly, and as expected, faults affected only performance.

Table 2. Configuration of Hardisc during simulations and synthesis.

Parameter	Number of Entries	Bits in Each Entry	Total Bits
Branch Target Buffer	16	19	304
Branch History Table	64	2	128
Jump Target Buffer	8	28	224
Buffer of Predictions	2	31	62
Return Address Stack	2	31	62
Instruction Fetch buffer ¹	4	36	144

¹ Two replicas in the protected architecture.

The results of proactive searching for faults in the register file with disabled memory latencies might be seen in Figure 7. The results are taken from the CoreMark simulation with faults enabled only in a register file group. With the selected fault-generation probability, a single fault was generated every 275 clock cycles, on average, during the whole simulation. The X-axis indicates a range of lifetimes of faults in terms of clock cycles. If the lifetime is

lower than 10 clock cycles, it is shown in the first column. If it is equal to or greater than 10 and lower than 20, it is shown in the second column, and so on. The Y-axis represents a count of faults with a lifetime described by the X-axis. The orange part indicates how many faults were solved by instruction masking. This means that the instruction rewrote the content of the faulty register with a new value. If the ACM corrected the fault, it is accounted for in the blue part. It can be seen that 93% of faults were resolved under 50 clock cycles from their creation. The ACM resolved more than 90% of all faults.

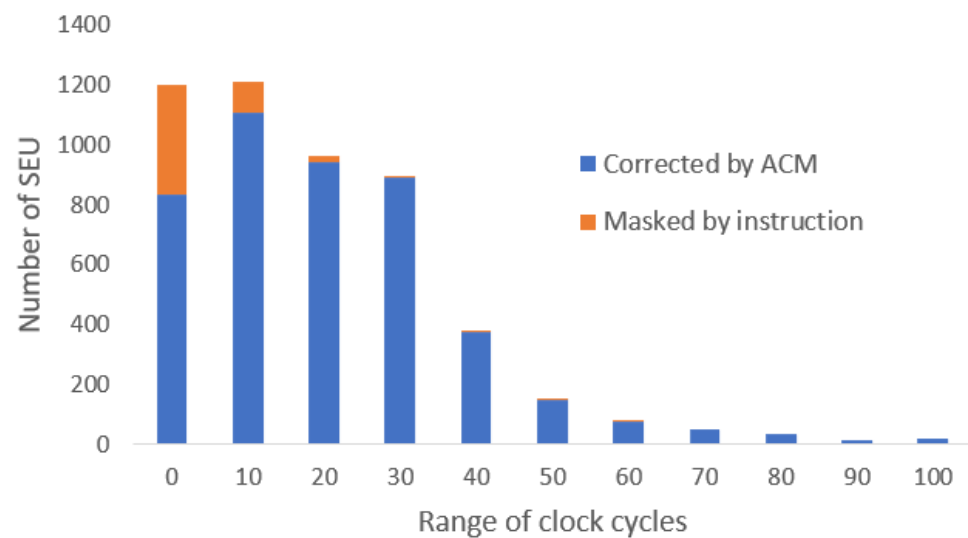


Figure 7. Lifetime of faults.

We also analyzed the impact of memory latencies on fault accumulation. In the simulation, where each memory access was extended randomly in the 1–5 clock cycles range, the masking was even less visible, and the ACM was more successful in finding faults. The outcome of this simulation was that 98% of faults were resolved under 50 clock cycles from their creation. This improvement is caused due to the behavior of the ACM when the OP-stage is empty or stalled. It leverages these clock cycles to proactively and systematically scan the register file. This means that even though the pipeline is stalled or empty, the register file remains protected.

6. Synthesis and Performance Results

6.1. Synthesis

We synthesized both the unprotected and protected versions of the Hardisc to get information on the protection scheme's impact on the maximal frequency, power consumption, and area of the core. The physical synthesis was performed in the Genus and Innovus by Cadence. The target technology was TSMC's 28 nm planar bulk CMOS. The synthesis was set up to disable optimizations, which could lead to merging or removing redundancy circuits. Since we synthesized only the core without memories, we had to provide input and output delays of the ports. Almost all ports of the core are interface signals for program and data buses. Input data ports were set to have an input delay of 75% of the target clock period, and the remaining input control ports had a delay of 50%. Output data and control ports had an output delay set to 50%. Table 3 summarizes the synthesis results of both versions under typical conditions (25 °C and 0.9 V).

The maximal frequency of the protected core is constrained by input/output paths. A path between two in-core registers with the highest delay has a 20% reserve for the clock period. So far, the development of the protected architecture has been focused mainly on reducing the overall area of the protected core and reducing the count of redundant elements, which helps to also reduce power consumption. The synthesis shows

that the inter-core logic has much potential in timing, so additional optimizations can be implemented to reduce the switching activity.

Table 3. Results from synthesis under typical conditions.

Parameter	Unit	Unprotected	Protected	Difference
Frequency	MHz	1364	1310	−3.93%
Sequential Instances	-	2960	4847	+63.75%
Combinatorial Instances	-	15,668	32,317	+106.26%
Total Area	mm ²	0.0268	0.0558	+107.97%
Leakage Power	μW	98	208	+112.25%
Dynamic Power ¹	μW	21,649	47,489	+119.36%
Total Power	μW	21,747	47,697	+119.33%

¹ The switching activity is sampled from one iteration of CoreMark at 1300 MHz.

6.2. Performance Assessment

To measure the performance of the Hardisc core, we have leveraged CoreMark and Dhrystone benchmarks. It is important to note that the implemented protection does not introduce any additional stalls or flushes in a faultless environment. This means that the performance is affected only slightly by a decrease in maximum frequency, so performance normalized at 1 MHz is the same for both versions of the core. Simulations showed that the performance of the core is 2.49 CoreMark/MHz and 1.73 Dhrystone/MHz. Table 4 is an extended version of Table 1 by the performance comparison and also includes protected and unprotected versions of Hardisc. Since this work focuses on a proposal and application of the protection, we are not providing a deeper performance evaluation.

Table 4. Comparison of processors with benchmarking.

Processor	Instruction Set	Protection Approaches ¹	Node Size [nm]	Frequency [MHz]	Performance	
					CoreMark	Dhrystone
AT697F [12,17]	SPARC-V8	TMR, RHBD, clock skew	180	100	203	-
GR740 [13,17,18]	SPARC-V8	RHBD, library	65	250	2248 ²	1700
TCLS [14,15]	Armv7-R	TCLS, library	65	311	-	520
SAMRH71 [19]	Armv7E-M	RHBP	150	100	504	-
RAD750 1st gen. [11]	PowerPC	RHBD	250	133	-	260
RAD750 2nd gen. [11]	PowerPC	RHBP	180	200	-	400
NOEL-V [20]	RISC-V	RHBD, library	28	800	3528 ³	-
Hardisc—protected	RISC-V	RHBD	28	1310	3262 ⁴	2266 ⁴
Hardisc—unprotected	RISC-V	RHBD	28	1364	3396 ⁴	2360 ⁴

¹ All processors/systems leverage EDAC to protect parts of the architecture/memories. ² Result for 4 threads from [17] (Table 7.3) and multiplied by 2.5, as the max. frequency is 250 MHz. ³ CoreMark data taken from dual-issue version and multiplied by provided max. frequency. ⁴ Compiled by GCC 11.1.0 with flags: `-O3 -march = rv32imc -mabi = ilp32 -funroll-loops`.

6.3. Comparison of Protection Approaches

A comparison of different protection approaches is shown in Table 5. The data are extracted from the available literature, and references are mentioned next to the name of the processors. We have analyzed the impact of the protection on the frequency, area, dynamic power consumption, and its additional requirements to the executing software, which affect the performance. The processors are grouped according to their general protection approach.

The TCLS [14] protection required three Cortex-R5 cores instead of one and an additional Assist Unit with an area of 14% of the single core. The authors declare that the frequency dropped by 10% compared to equivalent dual-core lockstep implementation. This means the frequency drop should be even more profound when compared to the single unprotected core. This TCLS approach requires a relatively long resynchronization

procedure after detecting a fault. The DCLS [22] approach to protecting the Cortex-A9 also requires a costly checker, which results in a higher increase of used resources. Results show that the core requires at least 26% more clock cycles to execute the test application due to additional required software routines. The approaches leveraging hardware multithreading [26,27] reduce the count of usable hardware threads by software, which has a high impact on performance. Sequential cells in a Cortex-R4 CPU [36] were replaced by TMR cells, and a clock skew was introduced for each flip-flop to protect the logic against SET. The caches were protected by the ECC. The results show that the frequency drops by 31%, and the chip area increases by 100%. The authors measured the area increase, including the caches. Without them, the relative area increase would be even higher. The dynamic power consumption increases by 104%. An area of the SHAKTI-F processor [28] increases by only 20%, but the frequency drops by 25%. Although the area increase is low, the processor contains the simplest instruction subset RV32I, and the study does not mention any prediction scheme or protection of CSR registers. It needs to be made clear whether the CSR registers are even implemented. Their protection would have a significant impact on the area increase.

The protection of Hardisc has a similar area increase to gate-level redundancy approaches and has the lowest impact on the maximum frequency. Another important factor is that the protection does not require additional software routines, which would affect the performance.

Table 5. Comparison of protection approaches.

Protection Approach	Processor	Drop in Frequency	Area Increase	Dynamic Power Increase	Requirements to a Software
System Lockstep	TCLS Cortex-R5 [14]	>10% ¹	214%	-	Resynchronization Checkpoints
	DCLS Cortex-A9 [22]	-	275% ^{2,3}	-	
Multithreading	fT03 [26,27]	-	16% ²	-	Fewer HW threads
	dfT03 [27]	-	25% ²	-	Fewer HW threads
Gate redundancy	TMR Cortex-R4 [36]	31%	>100%	104%	No
	LEON [37]	>8% ⁵	76% ⁴	-	No
Architecture	SHAKTI-F [28]	25%	20%	-	No
	Hardisc	4%	108%	119%	No

¹ Comparison to dual-core lockstep, the real drop should be even deeper. ² Synthesized into the FPGA, the area increase relates to the number of LUTs. ³ The checker IP is not protected. ⁴ Only the integer unit and register file are considered. ⁵ The additional clock skew (such as the one used in [12]) is not considered.

7. Discussion

We have proposed a protection scheme for processors against soft errors, which can be fully implemented at the RTL level using commercial technology libraries and process techniques. Although the protection is based on redundancy, the novelty lies in its application in the pipeline of the processor, which is separated into two sections. The first section only detects faults, and the second section masks them. So far, the protections have been implemented mainly at the system level by lockstep techniques or by using radiation-hardened libraries and processes. The advantages could be faster time to market, but these approaches usually have high prices and a significant negative impact on PPA data. There already exist radiation-hardened processors which can be implemented in commercial process techniques, but the primary protection is implemented at the technology library level, which contains protected gates with more transistors. Many radiation-hardened processors leverage error-correction codes to protect register files and caches, as the replication of these modules would have a high impact on PPA. The problem with such approaches is fault accumulation, which could lead to uncorrectable or even undetectable errors. Our implementation also uses error-correction code to protect the register file, but it also contains a proactive correction mechanism, which transparently scrubs the register file without

disrupting the running software. We have implemented the protection scheme into a RISC-V processor with the RV32IMC instruction set. The core code named Hardisc is described by SystemVerilog and also allows fault injection during RTL simulation. We have performed several fault-injection campaigns to test all implemented protection mechanisms. They prove that the protection scheme works as expected and can protect the core from unintended behavior. We also synthesized the core to 28 nm bulk technology to check the impact of the protection. It was shown that the protection lowers the maximal frequency only by 3.9% and increases power consumption by 119% and area by 108%. Critical paths were due to input/output delays of the ports, and the inter-core logic has more than 20% reserve in a clock period at the maximal frequency of 1310 MHz. So far, the core has not been optimized for switching activity, so we will leverage the reserve in the clock period to implement mechanisms to decrease it. The area and power consumption increases are similar to dual-lockstep and gate-level protection approaches, but the reaction time and ability to mask out faults are similar to TMR-protected architectures. Compared with other protections at RTL-level, it has a minimal impact on the maximum frequency. Another important fact is that it does not require the creation of backups or checkpoints in software, which would otherwise affect performance and require additional memory space. Future work will be focused on improving the dynamic power consumption and implementation of hardware multithreading.

Author Contributions: Conceptualization, J.M.; methodology, J.M.; software, J.M.; validation, J.M.; formal analysis, J.M.; investigation, J.M.; resources, J.M.; data curation, J.M.; writing—original draft preparation, J.M.; writing—review and editing, J.M. and L.K.; visualization, J.M.; supervision, L.K. and P.Č.; project administration, J.M.; funding acquisition, J.M. All authors have read and agreed to the published version of the manuscript.

Funding: The work reported here was supported by the Operational Programme Integrated Infrastructure for the project Advancing University Capacity and Competence in Research, Development and Innovation (ACCORD) (ITMS code: 313021X329), co-funded by the European Regional Development Fund (ERDF). This publication was also supported in part by the Slovak national project KEGA 025STU-4/2022, APVV-19-0401 and APVV-20-0346.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

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

References

1. Greb, K.; Pradhan, D. *Texas Instruments, Hercules™ Microcontrollers: Real-Time MCUs for Safety-Critical Products*; Texas Instruments: Dallas, TX, USA, 2011.
2. Avizienis, A.; Laprie, J.-C.; Randell, B.; Landwehr, C. Basic concepts and taxonomy of dependable and secure computing. *IEEE Trans. Dependable Secur. Comput.* **2004**, *1*, 11–33. [CrossRef]
3. Battezzati, N.; Sterpone, L.; Violante, M. *Reconfigurable Field Programmable Gate Arrays for Mission-Critical Applications*; Springer: New York, NY, USA, 2011. [CrossRef]
4. ECSS-Q-HB-60-02A; Techniques for Radiation Effects Mitigation in ASICs and FRGAs Handbook. ECSS Secretariat ESA-ESTEC Requirements & Standards Division: Noordwijk, The Netherlands, 2016. Available online: <http://microelectronics.esa.int/asic/ECSS-Q-HB-60-02A1September2016.pdf> (accessed on 23 April 2023).
5. Mavis, D.G.; Alexander, D.R. Employing radiation hardness by design techniques with commercial integrated circuit processes. In Proceedings of the 16th DASC. AIAA/IEEE Digital Avionics Systems Conference, Reflections to the Future, Irvine, CA, USA, 30 October 1997; pp. 15–22. [CrossRef]
6. Kobayashi, D. Scaling Trends of Digital Single-Event Effects: A Survey of SEU and SET Parameters and Comparison with Transistor Performance. *IEEE Trans. Nucl. Sci.* **2021**, *68*, 124–148. [CrossRef]
7. Ginosar, R. Ramon Chips, Ltd. Survey of Processors for Space, DASIA. 2012. Available online: https://docs.wixstatic.com/ugd/418640_087c23c99df24aa8acbf01b96dcd281a.pdf?index=true (accessed on 23 April 2023).
8. Abate, F.; Sterpone, L.; Lisboa, C.A.; Carro, L.; Violante, M. New Techniques for Improving the Performance of the Lockstep Architecture for SEEs Mitigation in FPGA Embedded Processors. *IEEE Trans. Nucl. Sci.* **2009**, *56*, 1992–2000. [CrossRef]

9. Marcinek, K.; Pleskacz, W.A. Variable Delayed Dual-Core Lockstep (VDCLS) Processor for Safety and Security Applications. *Electronics* **2023**, *12*, 464. [\[CrossRef\]](#)
10. Pontarelli, S.; Maestro, J.A.; Reviriego, P. Dependability Solutions. In *Dependable Multicore Architectures at Nanoscale*; Ottavi, M., Gizopoulos, D., Pontarelli, S., Eds.; Springer: Cham, Switzerland, 2018. [\[CrossRef\]](#)
11. Haddad, N.F.; Brown, R.D.; Ferguson, R.; Kelly, A.T.; Lawrence, R.K.; Pirkel, D.M.; Rodgers, J.C. Second generation (200MHz) RAD750 microprocessor radiation evaluation. In Proceedings of the 2011 12th European Conference on Radiation and Its Effects on Components and Systems, Seville, Spain, 19–23 September 2011; pp. 877–880. [\[CrossRef\]](#)
12. Atmel Corporation. Rad-Hard 32 bit SPARC V8 Processor AT697F, Rev. 7703E–AERO–08/11. 2011. Available online: <https://ww1.microchip.com/downloads/en/DeviceDoc/doc7703.pdf> (accessed on 23 April 2023).
13. Cobham Gaisler, A.B. GR740 Radiation Summary, Test Report, Doc. No. GR740-RADS-1-1-3, Issue 1.3. 2020. Available online: https://gaisler.com/doc/gr740/GR740-RADS-1-1-3_GR740_Radiation_Summary.pdf (accessed on 23 April 2023).
14. Iturbe, X.; Venu, B.; Jagst, J.; Ozer, E.; Harrod, P.; Turner, C.; Penton, J. Addressing Functional Safety Challenges in Autonomous Vehicles with the Arm TCL S Architecture. *IEEE Des. Test* **2018**, *35*, 7–14. [\[CrossRef\]](#)
15. Iturbe, X.; Venu, B.; Ozer, E.; Poupat, J.-L.; Gimenez, G.; Zurek, H.-U. The Arm Triple Core Lock-Step (TCLS) Processor. *ACM Trans. Comput. Syst.* **2019**, *36*, 1–30. [\[CrossRef\]](#)
16. Arm Limited. *Arm[®] Cortex[®]-A76AE Core Technical Reference Manual*; Revision: r1p1, Issue 00; Arm Limited: Cambridge, UK, 2022.
17. Daněk, M.; daiteq s.r.o. Benchmark Specifications and Report, 2021-IETR021ESAIP013-V1.7. 2021. Available online: http://microelectronics.esa.int/riscv/Benchmark_specification_v1.7.pdf (accessed on 23 April 2023).
18. CAES. GR740 Next Generation Microprocessor Flight Models, TEC-ED & TEC-SW Final Presentation Day. June 2021. Available online: <http://microelectronics.esa.int/finalreport/GR740-NGMP-FinalPresentation-Public-2021-06-01.pdf> (accessed on 23 April 2023).
19. Microchip Technology Inc. *SAMRH71 Rad-Hard 32-Bit Arm[®] Cortex[®]-M7 Microcontroller for Aerospace Applications, Complete Data Sheet, DS60001593H*; Microchip Technology Inc.: Chandler, AZ, USA, 2022.
20. CAES. Gaisler NOEL-V SoC Applications and Ecosystem, RISC-V in Space 2022. Available online: http://microelectronics.esa.int/riscv/rvws2022/presentations/06_ESA_RISC-V_in_Space-NOEL-V.pdf (accessed on 23 April 2023).
21. Fazeli, M.; Namazi, A.; Miremadi, S.G. Robust Register Caching: An Energy-Efficient Circuit-Level Technique to Combat Soft Errors in Embedded Processors. *IEEE Trans. Device Mater. Reliab.* **2010**, *10*, 208–221. [\[CrossRef\]](#)
22. de Oliveira, A.B.; Rodrigues, G.S.; Kastensmidt, F.L.; Added, N.; Macchione, E.L.A.; Aguiar, V.A.P.; Medina, N.H.; Silveira, M.A.G. Lockstep Dual-Core ARM A9: Implementation and Resilience Analysis Under Heavy Ion-Induced Soft Errors. *IEEE Trans. Nucl. Sci.* **2018**, *65*, 1783–1790. [\[CrossRef\]](#)
23. Doran, H.D.; Lang, T. Dynamic Lockstep Processors for Applications with Functional Safety Relevance. In Proceedings of the 2021 26th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), Vasteras, Sweden, 7–10 September 2021; pp. 1–4. [\[CrossRef\]](#)
24. LaFrieda, C.; Ipek, E.; Martinez, J.F.; Manohar, R. Utilizing Dynamically Coupled Cores to Form a Resilient Chip Multiprocessor. In Proceedings of the 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'07), Edinburgh, UK, 25–28 June 2007; pp. 317–326. [\[CrossRef\]](#)
25. Sim, M.T.; Zhuang, Y. A Dual Lockstep Processor System-on-a-Chip for Fast Error Recovery in Safety-Critical Applications. In Proceedings of the IECON 2020 The 46th Annual Conference of the IEEE Industrial Electronics Society, Singapore, 18–21 October 2020; pp. 2231–2238. [\[CrossRef\]](#)
26. 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. [\[CrossRef\]](#)
27. 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. [\[CrossRef\]](#)
28. Gupta, S.; Gala, N.; Madhusudan, G.S.; Kamakoti, V. SHAKTI-F: A Fault Tolerant Microprocessor Architecture. In Proceedings of the 2015 IEEE 24th Asian Test Symposium (ATS), Mumbai, India, 22–25 November 2015; pp. 163–168. [\[CrossRef\]](#)
29. RISC-V Foundation. *The RISC-V Instruction Set Manual, Volume I: User-Level ISA*; Document Version, 20191213; Waterman, A., Asanović, K., Eds.; EECE Department, University of California: Berkeley, CA, USA, 2019.
30. Santos, D.A.; Luza, L.M.; Zeferino, C.A.; Dillio, L.; Melo, D.R. A Low-Cost Fault-Tolerant RISC-V Processor for Space Systems. In Proceedings of the 2020 15th Design & Technology of Integrated Systems in Nanoscale Era (DTIS), Marrakech, Morocco, 1–3 April 2020; pp. 1–5. [\[CrossRef\]](#)
31. ARM Limited. *IHI 0033A, AMBA[®] 3 AHB-Lite Protocol v1.0, Specification*; ARM Limited: Cambridge, UK, 2006.
32. Riasanovsky, N. Understanding RISC-V Calling Convention. Available online: https://inst.eecs.berkeley.edu/~cs61c/resources/RISCV_Calling_Convention.pdf (accessed on 23 April 2023).
33. RISCV-DV Random Instruction Generator. Available online: <https://github.com/google/riscv-dv> (accessed on 23 April 2023).
34. Spike RISC-V ISA Simulator. Available online: <https://github.com/riscv-software-src/riscv-isa-sim> (accessed on 23 April 2023).
35. EEMBC. CoreMark. Available online: <https://www.eembc.org/coremark/> (accessed on 23 April 2023).

36. Özer, E.; Bull, D.M. SEU and SET-tolerant ARM Cortex-R4 CPU for Space and Avionics Applications. In Proceedings of the Second Workshop on Manufacturable and Dependable Multicore Architectures at Nanoscale (MEDIAN'13), Avignon, France, 30–31 May 2013.
37. Gaisler, J. European Space Agency, A Portable and Fault-Tolerant Microprocessor Based on the SPARC V8 Architecture. In Proceedings of the International Conference on Dependable Systems and Networks (DSN'02), Washington, DC, USA, 23–26 June 2002. [[CrossRef](#)]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.