Improving the Robustness of Redundant Execution with Register File Randomization

Ilya Tuzov, Pablo Andreu, Laura Medina, Tomas Picornell,
Antonio Robles, Pedro Lopez, Jose Flich and Carles Hernández
DISCA, Universitat Politècnica de València, Campus de Vera s/n, 46022, Spain
tuil@disca.upv.es, pabance@upv.es, laumecha@upv.es, tompic@gap.upv.es,
arobles@disca.upv.es, plopez@disca.upv.es, jflich@disca.upv.es, carherlu@upv.es

***Abstract*—Staggered Redundant execution (SRE) is a fault-tolerance mechanism that has been widely deployed in the context of safety-critical applications. SRE not only protects the system in the presence of faults but also helps relaxing safety requirements of individual elements. However, in this paper, we show that SRE does not effectively protect the system against a wide range of faults and thus, new mechanisms to increase the diversity of homogeneous cores are needed. In this paper, we propose Register File Randomization (RFR), a low-cost diversity mechanism that significantly increases the robustness of homogeneous multicores in front of common-cause faults (CCFs) and register file wearout. Our results show that RFR completely removes the failure rate for register file CCFs for certain workloads and reduces by a factor of 5X the impact of stress related register file aging for the workloads analysed. Our implementation requires less than 50 RTL lines of code and the area (FPGA logic) overhead of RFR is less than 0.2% of a 64-bit RISC-V core FPGA implementation.**

***Keywords*—RISC-V, common cause failures, staggering, aging**

I. Introduction

Multicore platforms are usually the primary option to cope with the huge computational power demands of fully autonomous safety-critical systems. However, these high-complexity computing platforms find difficulties to achieve the most stringent certification requirements imposed by functional safety standards [1]. Achieving safety certification requires ensuring functional and temporal correctness in the most stringent yet plausible situations.

Redundant execution (RE) is a fault-tolerant mechanism that has been widely deployed in the context of safety-critical applications. RE not only protects the system in the presence of faults but also helps relaxing safety requirements of individual elements. For the latter, some form of diversity has to be implemented in the multicore to meet independence requirements. Staggered Redundant execution (SRE) is the most common approach to achieve diversity. SRE can be implemented by hardware or software means. In hardware, staggering is usually implemented using lockstep execution. Examples of lockstep processors are the ST microelectronics SPC56XL70 [2] and the Tricore [3] chips. In COTS multicores, SRE can be regarded as inherent property in systems of very high-complexity [4] or can be achieved by appropriately delaying and monitoring the execution of redundant tasks [5].

Unfortunately, as we show in this paper, SRE does not protect the system in front of common-cause faults (CCFs) that do not have a transient nature. For instance, faults that are a consequence of manufacturing defects that escape the diagnostic coverage of in place safety mechanisms or faults that originate due to the wearout of the devices are not effectively protected with SRE. Thus, new means are required to increase the diversity in homogeneously designed computing devices. In this context, we propose a new safety mechanisms that improves the robustness of redundant execution in multicore platforms. The proposed mechanism performs a Register File Randomization (RFR) that increases the robustness of homogeneous multicores by improving the protection against CCFs affecting the register-file. In particular, we make the following contributions:

- We design the RFR mechanism and define how to design efficient randomizer circuits.
- We tailor the RFR mechanism to a particular processor design and implement it in a RTL description of this 64-bit RISC-V core CPU.
- We analyse the effectiveness of SRE in front of CCFs originated in the register file of a 64-bit RISC-V core pipeline in a triple-modular redundancy (TMR) setup.
- We evaluate the effectiveness of RFR to protect the system from CCFs affecting the register file, the benefits provide by RFR in the wearout of the register file, and the overheads incurred by the proposed implementation.

The rest of the paper is organized as follows. Section II introduces the necessary background on safety critical systems and relevant fault models. Section III describes our RFR proposal. RFR is evaluated in Section IV. Finally, Section V surveys the related research and conclusions are drawn in Section VI.

II. Background

A. Functional Safety Certification Challenges

Safety-critical systems have to go through a certification process to show they adhere to the requirements imposed by functional safety standards. Functional safety standards exist for different domain applications and impose different constraints to the development of electronic equipment. In the context of autonomous applications, the more relevant safety standards are the Road vehicles – Functional safety

(ISO26262 [6]) and the Functional Safety of Electrical/Electronic/Programmable Electronic Safety-related Systems (IEC-61508) [7].

Both ISO26262 and IEC-61508 define safety integrity levels (SILs or Automotive SILs for ISO26262) representing the risk that can be associated to a specific system functionality. The higher the SIL the more measures have to be taken to avoid the possibility of hazardous situations. Enabling safety-relevant systems to operate autonomously requires the core functionalities of those systems to achieve the highest SIL [8] because autonomy, at least in transportation systems, generally defeats the possibility of transitioning to a safe state. The implications of this in the design of autonomous systems is huge, since complexity has been usually avoided in critical applications [9].

B. SIL Decomposition

Achieving ASIL-D –assuming ISO26262 nomenclature– requires reaching diagnostic coverage values and failure rates that are generally beyond of what can be achieved for complex CPU cores with state of the practice tools and methodologies [10]. Thus, to achieve the highest ASILs (e.g., ASIL-D or ASIL-C), practitioners rely on the properties of SIL decomposition [10]. SIL decomposition is a mechanism that allows to split the functionalities of a system and attribute each of them a given criticality level. To apply SIL decomposition functionalities with a given SIL must be independent. With SIL decomposition, one can relax criticality requirements of individual CPU cores employing redundancy if these cores are shown to operate on an independent manner. In practical terms, independent functioning translates into the ability to exhibit different behaviour (a.k.a diversity) in presence of common-cause faults (CCFs). CCFs are those faults that can affect replicas of a system similarly and have the potential to cause erroneous outputs of a circuit replica to get undetected.

C. Diverse Redundancy

Truly diverse CPU cores can only be achieved when these elements are designed on an independent manner. For instance, cores are diverse when using different ISAs or completely different microarchitectural implementations. However, in the majority of systems, the cost of developing, verifying, and validating such electronics equipment devices is not negligible. The cost of implementing a fully diverse CPU core roughly multiplies by N the development costs of a N-modular redundancy system. Therefore, designing fully diverse CPU cores becomes unpractical.

The most common approach to implement diversity is lockstep execution. Dual-core lockstep (DCLS) processors are widely deployed in automotive applications and have been shown compatible with ASIL-D requirements in certain applications [3], [2]. In DCLS sytems, the leading and trailing cores will, by construction, never execute the same instructions at the same time. Thus, transient common-cause faults (e.g., voltage drops) affecting the system will not produce the same erroneous behaviour in each of the replicas. Additionally, in the physical implementation process, the layout of these functionally identical cores can be slightly modified (e.g., implementing rotation) to increase the coverage of faults related to the physical implementation.

D. Fault Models

As explained above, DCLS provides diversity for transient faults and has the potential to tolerate certain types of manufacturing defects when implementing diversity at the physical implementation level too [11]. However, for the latter it is not clear to what extent the current diversity techniques will be effective in future manufacturing processes targeting smaller technology nodes. Note that for complex systems incorporating small technology nodes, the lack of defect models data, the coverage limits of testing [12], and the amplified impact of aging [13] pose serious difficulties to the reliability modeling of such systems.

In general, as the complexity of cores increases and node technology shrink down, the effectiveness of the diversity means provided by DCLS systems might not suffice [14]. Thus, new techniques increasing the diversity of the CPU cores are required. These new techniques must consider not only random transient fault effects but also systematic effects that appear repeatedly in the same device structures [15] and the correlation effects occurring in neighbouring cells [16].

CCFs affecting register file structures are especially critical for the robustness of multi-core systems, as it is evinced in [17]. Despite register files occupy relatively small area in modern processors, they have very high access rate, and their faults are rapidly propagated across the whole system [18].

III. Register File Randomization

In this section we describe the Register File Randomization (RFR) mechanism proposed in this paper, its implementation in a RTL RISC-V core description, and the increased robustness properties it provides.

A. Architectural Modifications

RFR is a low-cost technique that increases the robustness of SRE. The goal of this technique is to dynamically modify the physical register file access pattern by using a randomization mechanism. We distinguish the architectural (or logical) from the physical register file entry. This randomization mechanism allows us to have different register file mappings (i.e modifying the binding from the logical register to the the particular physical register entry) so that systematic faults affecting the register file structures can be detected with SRE. In particular, RFR combines the index bits required to access the register file with the core ID and a random value to generate a randomized index to access register file contents.

RFR modifies the access to the register file by keeping the remaining parts of the core unaltered. These modifications are needed in each of the SoC cores. The architectural view of the RFR mechanism is depicted in Figure 1. As shown in the plot, RFR does only require the introduction of a hashing circuit at the register file input to randomize the indexes to access

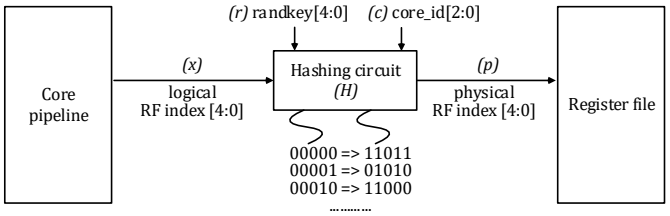

Fig. 1. Architectural modifications needed to implement RFR

the register file. Additionally, RFR needs to have access to the core ID bits, which are usually available at several core structures, and requires having a random number generator able to provide a new key value every time the system is reset. The rest of the core remains unaltered. Random number generators are already included in several SoC designs [19] or can be added to the SoC at a very low cost [20]. Another possibility is to compute the random value as part of the boot up process. Note that, in our case, a pseudo random number generator is sufficient to achieve the randomization properties needed by RFR.

B. Randomizer Design

The hashing function to randomize the register index must fullfill the following requirements:

- (R1) The resulting register mapping (randomization) must be different at each core in the SoC (i.e. the same architectural register is mapped to a different physical register at each core).
- (R2) Every physical register is assigned to one and only one architectural register (i.e. there are no collisions in the mapping).
- (R3) At a given core, each physical register can be chosen for mapping with equal probability.

R1 is required to ensure diversity and it is fullfilled if the hash function takes into account not only the architectural register index but also a different key at each core. This can be easily accomplished with a key computed as a function of core ID.

R2 is mandatory, as mapping several architectural registers to the same physical register is obviously incorrect. It is fullfilled if the hash function maps each element of the output set to exactly one element of the input set (i.e, the hash function is injective). As the cardinal of the input and output set is the same (i.e. the register index has the same number of bits), the hash function should be bijective.

R3 guarantees that the utilization of the registers is as uniform as possible during the lifetime of the device. It is fulfilled if the hash function considers a random key as an input.

Formally, the hashing function H obtains the physical register p from three inputs: the index of the architectural register x, a random key value r, and the core ID c.

Different valid hashing functions could be used. For the sake of implementation simplicity, we propose using a hashing function based on the bitwise XOR function. Bitwise XOR has been widely used for hashing, reducing conflict misses in caches [21] , branch predictors [22] and routing algorithms [23]. In particular, the hashing function obtains the physical register p by bitwise xoring the index of the architectural register x with a key value k:

$$p = H(x, k) = x \oplus k$$

The key value k depends on a random key value r and the core ID c, and is computed by using also a bitwise XOR, as follows:

$$k = r \oplus D(c)$$

where D is a function that computes a hash key as function of core ID. This function could be as simple as the identity. However, we propose using a function that tries to maximize the Hamming distance among computed keys. This enforces diversity and randomization. By using this approach, we minimize the possibility of not capturing systematic error patterns in neighbouring or adjacent registers [16]. It is noteworthy to mention that, as the number of cores in the system increases, the effect of randomization decreases (due to the reduced Hamming distance among the keys). However, this can be easily solved by clustering groups of cores in a manycore, and reusing IDs between them, since redundancy is usually employed between a limited number of cores (e.g., three modular redundant system).

The random key r is computed every time the system is reset and is the same for all cores in the SoC. Using a different key value after each reset guarantees R3. On the other hand, the key value $k = r \oplus D(c)$ leads to a different value at each core, as r is the same for all cores but $D(c)$ is obviously different at each core, thus fulfilling R1. Finally, the physical register is computed by using a bitwise XOR function ($p = x \oplus k$), which is bijective, thus fulfilling R2.

Notice that the number of bits of the register index p and x, the random key value r and the result of $D(c)$ (c is the core ID) should be the same. In practice, this means that the number of architectural registers will determine the number of bits to use. With the 32 typical register file size, 5 bits will be used. Therefore, a 5-bit random key is required and, most important, $D(c)$ range is limited to 32. This limit should not be a problem, since, as explained above, the amount of cores used to implement redundancy is limited. For large SoCs, cores can be clustered into smaller groups, and IDs reused using the least significant bits of the actual core ID.

Figure 2 shows a detailed view of the randomizer circuit. As shown in the plot, the implementation is very simple. The random key value r is computed once per system reset and the $D(c)$ function is fixed. Therefore, the key k could be precomputed and stored in a system special purpose register. In addition, this key is bitwise xored with the logical register index x to obtain the physical register index p. At the end, only a two-input XOR gate is required at the address input of the register file. Although this may lead to a slight increase (a 2-input XOR gate delay) in the register file access time, it

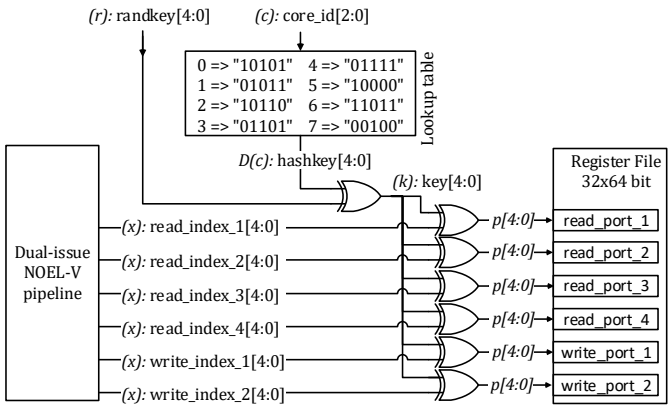

Fig. 2. Hash circuit design. Inputs are architectural register Index, core ID and a random key

should not be a problem unless it is in the critical path, which is not usually the case. Notice, though, that the implementation shown corresponds to a 6-port register file (see below).

The RFR (randomizer) circuit itself can be protected against faults by means of the common TMR scheme. Nevertheless, given that RFR is deployed in a system that already implements a diverse redundancy (SRE), such additional protection of RFR circuit would be usually unnecessary.

C. Robustness Improvement

RFR improves the robustness in two main aspects. First, it allows to correct or detect common-cause faults originated in the register file by modifying the effective physical location of registers in the different core replicas. Notice that common-cause faults are more likely to affect the same physical register since core replicas are implemented using the same SRAM macrocells and usually have the same layout structure. Additionally, in the absence of RFR, the same register entries are stressed with very similar activity patterns. Specially, when they are devoted to implement SRE.

Second, RFR also mitigates the effect of wearout in the layout due to stress by uniformizing the utilization of register file cells. Generally, Application Binary Interface (ABI) and programs impose specific usages to registers that make them to be unevenly utilized. Additionally, in the context of critical systems, workloads are usually repetitive, which exacerbates more this uneven distribution effect. By using the RFR technique proposed in this paper, all registers are used uniformly, which reduces the impact of aging effects that have a direct dependency on the utilization, such as the hot carrier injection (HCI) [24] or time-dependent dielectric breakdown (TDDB) [25].

D. RFR implementation in the NOEL-V SoC

We have implemented RFR in the NOEL-V RV64 core to validate the mechanism properties and the possibility to bring it to a real design. NOEL-V is a processor designed by Cobham Gaisler. The RTL description of the NOEL-V core is open-source (GPLv2) and can be downloaded from the Cobham Gaisler website [26]. The 64-bit RISC-V ISA defines 32 integer and general purpose 64-bit registers and 32 64-bit floating point registers. Thus, 5 bits are required to index both the general purpose integer and floating point register files.

On the other hand, NOEL-V core uses a 7-stage pipeline. Register file access is located at the 3rd stage of pipeline. As stages 1^{st} and 5^{th} are devoted to instruction and data cache access, respectively, they will incur in a higher delay and these stages will likely to be the in critical path. Thus, the slight increase in register access time due to the RFR should not have any impact in clock cycle. Indeed, as the core is dual-issue, up to 4 registers can be read and 2 registers can be written every clock cycle. This means that the RFR mechanism must be applied on a per read and write port basis. However, the circuitry related to the computation of the key ($k = r \oplus D(c)$)) will be shared among all ports.

Figure 2 shows the implementation of RFR in NOEL-V. To implement RFR we take the core ID bits (c) from *hindex* parameter (a read-only register). For the random 5-bit key (r) we have implemented a pseudo random number generator (PRNG) that uses a linear-feedback shift register as described in [20]. The bits of the PRNG are kept constant and only modified after a system reset to ensure the consistency of register file data. The hash obtained from the core ID ($D(c)$) is bitwise xored with the random key (r) to obtain the key value k bits. Thus, the k key is computed once after system reset and it is stored in a special purpose register. This value is bitwise xored with the architectural register index x to obtain the physical index p. As the register file has 4 read and 2 write ports, randomization has to be performed at each address port.

An special case arises when applying the RFR mechanism to RISC-V architecture. RISC-V (and other) ISA specification defines one architectural register to be read-only and hardwired to zero. In particular, the "x0" is a dedicated $zero$ register. We can deal with this issue in two ways. First, we can ignore that special case at the RFR mechanism, thus putting the complexity in the core pipeline and register file design. The register file must be aware of which index maps to register x0 to avoid writes and keep the guaranteed zero value.

Another solution is to consider the special mapping at the RFR level. The hashing function should do nothing when applied to x0 (i.e, architectural register 00000 will be mapped to physical register 00000). As stated above, as the hash function must be bijective, this implies that the architectural register that the hash function originally map to 00000 should now return a different value. The only valid (or available) one is precisely the value the mapping originally returned to architectural register x0. Formally, to deal with this case, the hash function is defined as follows:

$$p = H(x,k) = \begin{cases} 0 & \text{if } x = 0 \\ k & \text{if } x = k \\ x \oplus k & \text{otherwise} \end{cases}$$

Notice that this solution, while simpler, has the problem that one architectural register (i.e. register with index k) will not be randomized, thus only partially fulfilling R1 and R3

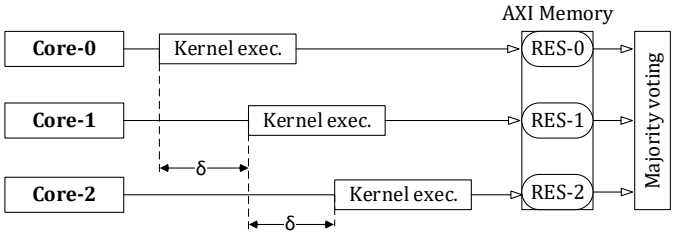

Fig. 3. NOEL-V multicore assembly under study

TABLE I. Details of the considered workloads

Workload	Description	Duration under δ=0 (clock cycles)
Matmult	Matrix multiplication adapted from MiBench-automotive suite	3800
AES	AES-256 encryption adapted from Tiny-AES	18800
Dijkstra	Finding shortest paths on the graph Adapted from Mibench-networking suite	12400
CRC	Cyclic redundancy check from Malardalen WCET suite	1500
Qsort	Quick sort based on stdlib	15000
BinarySearch	Binary search within an array of key-value structures Heavily modified version of bs from Malardalen WCET suite	3400
FIR	Finite impulse response filter adapted from Malardalen WCET suite without major modifications	3600

requirements. In this paper, we have opt for this simple solution. This has the advantage of avoiding the need to introduce deeper modifications to the RTL code which would require re-verification of the whole integer pipeline.

The NOEL-V core amounts a total of 12473 lines of code whereas our modifications, without considering the PRNG external module, incur in only 46 lines of code concentrated in only 2 files. The PRNG is a simple module and in our particular implementation we required only 22 lines of code.

IV. Experimental Evaluation

A. Platform Setup

As stated above, we have implemented the proposed RFR mechanism in the NOEL-V [26] processor from Cobham Gaisler. NOEL-V is an open-source core that implements a 64-bit RISC-V core [27] with the integer, floating point, atomics and multiply and divide extensions (a.k.a IMAFD).

To evaluate our proposal, we use the multicore configuration of NOEL-V to implement a triple-modular redundant system. Diverse redundancy in this system is achieved by means of staggered execution (using only software means). As workloads we have selected seven benchmark programs detailed in Table I. These benchmarks are adapted from MiBench [28], Malardalen WCET [29] and Tiny-AES [30] suites to operate in a multicore configuration in which each benchmark is executed in three cores simultaneously. The execution of the workload (kernel) in each core can be delayed by a configurable number of clock cycles proportional to core ID ($\delta \times c$). In this case study, three different inter-core delays have been tested under $\delta = 0$, $\delta = 100$ and $\delta = 1000$. The processing results of each workload are stored in the form of a linear array in the dedicated area of AXI memory. Figure 3 shows further details regarding the TMR design under study.

To evaluate the robustness improvement attainable by means of proposed RFR mechanism we perform two different experiments.

The first experiment relies on simulation-based fault injection (SBFI) to evaluate the robustness improvement of the design under test (DUT) in front of CCFs. Using custom publicly available SBFI tool [31] we performed 5000 SBFI tests per each combination of seven considered workloads with three different staggering delays δ.

The faultload is represented by transient faults (bit-flips) simultaneously affecting register files of three NOELV cores. Bit-flips are uniformly sampled in time (any time instant during workload execution) and space (any of 32x64 Register File (RF) bits). Each fault sampled for one of the cores is simultaneously injected into the same RF bit of the rest of NOEL-V cores. The classification of experimental outcome in each SBFI test comprises two steps. First, the fault effect is determined for each individual NOEL-V core, being it either a *match* (masked or latent fault) if the core has stored the correct results to its dedicated memory area, or a *failure* otherwise (incorrect/absent results). Second, the results produced by each core are voted attending to the TMR scheme, and compared to the reference results (obtained during fault-free simulation), being registered as either *TMR match*, or *TMR failure*.

For the comparison purposes, the same SBFI test has been also performed for the DUT with disabled RFR mechanism. SBFI experiments have been executed on a HPC cluster, using 150 parallel simulation processes. The total execution time to perform the scheduled 210 thousands of SBFI runs amounted to roughly 50 hours.

The second test performs simulation-based profiling of RF accesses in each NOEL-V core to evaluate the uniformity of registers utilization under the enabled and disabled RFR mechanism. As previously explained, this uniformity is considered as one of the indicators of RF wearout intensity. In each of 1000 tests, a new random key is generated, the workloads are simulated and the number of read and write accesses to each register is traced. Remember that each RF has four read ports and two write ports, since NOEL-V implements dual-issue pipeline. For the simplicity of subsequent analysis, all accesses to these ports are aggregated on a per-register basis, i.e. producing a *number of reads* and *number of writes* metric for each physical register.

B. Robustness Evaluation

The results of SBFI experiment for the original DUT (with disabled RFR mechanism) are summarized in Fig. 4-a. As it can be seen, the failure rate of each individual NOEL-V core ranges between 1% (CRC workload) and 6% (Qsort workload). An important observation regarding the original model is that staggered execution provided rather marginal robustness improvement for the TMR assembly. Indeed, a noticeable reduction of TMR failure rate (in comparison with individual cores) is observed only in four cases (out of 21): Matmult

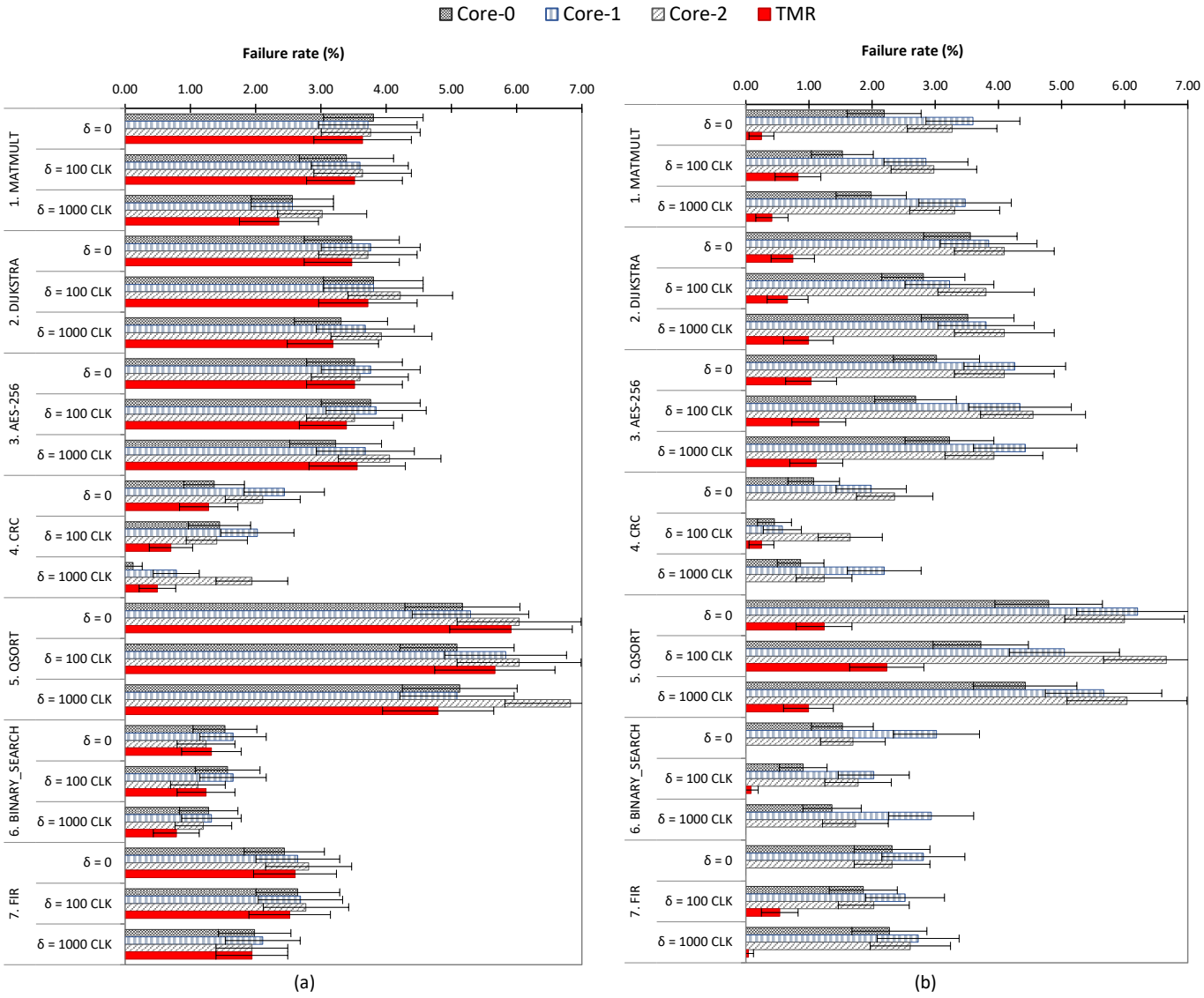

Fig. 4. Failure rate of individual NOEL cores and TMR assembly under staggered execution: (a) register randomization disabled, (b) register randomization enabled. Error bars represent sampling error for 95% confidence.

with high inter-core delay ($\delta = 1000$), CRC with medium and high delay ($\delta = 100$ and $\delta = 1000$), and Binary_Search with high inter-core delay. These results indicate that staggered execution is rather inefficient for protecting the DUT against CCFs in the register file, even under the increasing inter-core delays. This may be explained by the very high criticality of some RF registers, i.e. bit-flips affecting these registers cause a processor core to fail independently of fault injection time.

The results of SBFI experiment for the DUT protected by RFR mechanism are summarized in Fig. 4-b. As it can be seen, enabling RFR leads to significant reduction of TMR failure rate in all cases. More precisely, this robustness gain ranges between 2X (Qsort) and 10X (Matmult, Binary search). Furthermore, in some cases, the TMR was able to tolerate all injected CCFs. In particular, no TMR failures has been observed in five cases: CRC ($\delta = 0$, $\delta = 1000$), Binary_Search ($\delta = 0$, $\delta = 1000$), FIR ($\delta = 0$). This robustness improvement can be explained by mapping the critical logical registers to different physical registers in each core, which lowers the probability of CCF to cause simultaneous failure of several NOEL-V cores.

The results of RF profiling experiment are summarized in Fig. 5. As it can be seen from Fig. 5-a, in the original DUT six consecutive registers (*x11* to *x16*) have dominating access rate, which exceeds the access rate of the rest of registers by an order of magnitude. In the RISC-V ABI, this range of registers corresponds to function arguments. This uneven distribution of registers utilization is especially pronounced in case of write transactions. The peak access rate (8 millions of read/write transactions in 1000 consecutive workload executions) is measured for *x14* and *x15*. While the access rate of *x3* and x4 is close to zero. In the RISC-

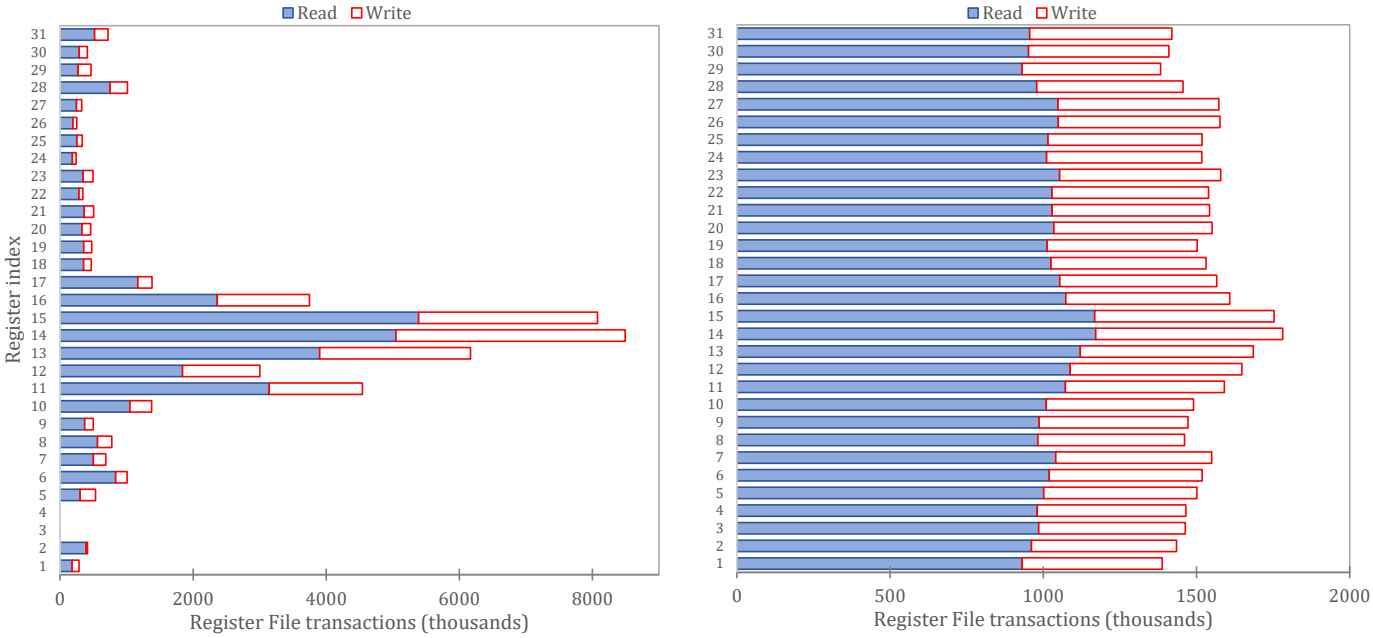

Fig. 5. Amount of read and write operations (total of all tested workloads) measured for each RF register in a sequence of 1000 tests: (a) RF randomization disabled, (b) RF randomization enabled

V ABI, these registers are the global and the thread pointer, respectively.

As it can be seen from the Fig. 5-b, after enabling the RFR mechanism, the RF accesses become distributed much more evenly. The resulting access rate of all registers ranges between 1.4 and 1.7 millions of read/write transactions per 1000 workload executions (remember that a new random key is generated in each execution). In such a way, RFR mechanism not only equalizes the the relative criticality of physical RF registers, but also reduces the absolute stress suffered by RF, as the maximum access rate per register is reduced from 8×10^6 per 1000 workload runs to just 1.7×10^6.

Finally, in the original system some logical registers have much higher access rates than the rest of the registers (as shown in Fig. 5-a), which leads to increased electrical stress of corresponding physical registers. In the instrumented system (with RFR mechanism) these logical registers periodically become remapped onto different physical registers, allowing to equalize the electrical stress suffered by each register, and thus to extend the overall device lifetime.

Fig. 6 illustrates the relative lifetime improvement, computed as the ratio between the access rate of most active register in the original system, and the access rate of most active register in the RFR-instrumented system. As it can be seen, in our case study, enabling RFR reduces the stress of physical registers (improves RF lifetime) from 4.7 times (CRC) to 7.4 times (binary search).

C. Overhead Analysis

Both original and RFR-instrumented NOEL-V designs have been synthesized and implemented (as a part of a custom SoC) onto a Virtex UltraScale+ FPGA (device part xcvu9pflga2104-2L), by means of Vivado 2019.1 suite. The SoC is composed of 4 cores (although our TMR system only uses 3 cores). According to the logic utilization and timing reports generated by Vivado, our implementation of RFR doesn't introduce noticeable overhead neither on utilization of FPGA resources, nor on timing properties (critical paths).

The precise utilization of LUTs in original and RFR-instrumented SoC is listed in Table II. As it can be seen, the difference between the FPGA resources employed by each core using the same implementation is higher than the

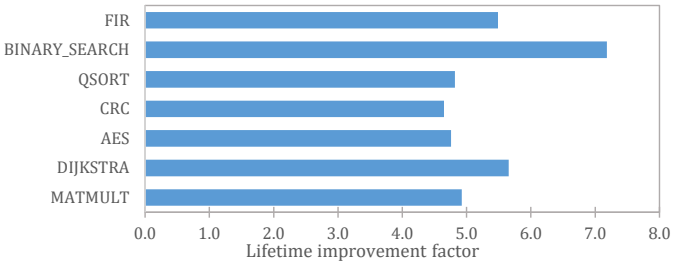

Fig. 6. Lifetime improvement factor resulting from equalized FR utilization

TABLE II. Utilization of LUTs in the original and RFR-instrumented SoC

	Core-0	Core-1	Core-2	Core-3	SoC
Original SoC	17501	17915	17889	18150	198536
RFR-instrumented SoC	17576	17164	18623	17786	198865

area overheads of RFR which makes difficult to quantify its (negligible) impact on area overheads. These minor differences in used FPGA resources among the cores can be attributed to logic optimizations [32] applied by Vivado on cross-boundary basis, such as resource sharing, LUT combining, retiming, etc. On the other hand, the entire RFR-instrumented SoC utilizes 329 more LUTs than the original SoC, which translates into less than 0.2% of utilization overhead.

The area overheads of a potential ASIC implementation of RFR can also be regarded as negligible. The RFR hashing circuit for the considered DUT would require the following macrocells: 60 XNOR gates, 12 AND gates, 30 MUX2 gates, 35 XOR gates, and 15 Flip-Flops. Remember that we remove the delay of xoring the random key and the hash function of the core ID by storing the result in a dedicated register. Thus, the resulting delay of the hashing circuit would be equal to a chain of three logic gates: one XNOR, one AND, and one MUX2.

V. Related Work

Register file errors can be supported by using error correction codes (ECC) or parity. The Intel Montecito [33] implements parity in the register file. Implementing parity incurs in little overheads, but requires software support to recover from errors. On the contrary, the utilization of ECC involves higher costs, but enables the automatic correction of faults. In the context of processors operating in harsh environments (e.g., the space domain), Single Error Correction Dual Error Detection (SEC-DED) can be applied to protect the register file contents. This is the case of the LEON SPARC core implemented in the NGMP multicore CPU [34], the Jetson NVIDIA platform or the Infineon AURIX processor [3]. Unfortunately, the simple protection of memory structures and interconnects is not enough to achieve the stringent robustness requirements imposed by the highest criticality fail-operational applications (e.g ASIL-D autonomous systems). In these very complex systems, dual or triple modular diverse redundancy is additionally required to achieve the target failure metrics imposed by domain-specific safety standards [8].

Several works have been proposed to achieve diverse redundant execution in multicores. Lockstep multicore designs implementing the sphere of replication at the on-chip bus (a.k.a light-lockstep) are very well known and have been deployed [3], [2] or proposed [11] in the context of automotive ASIL-D and ASIL-C applications. For COTS platforms, software solutions to achieve diverse redundant execution have also been proposed for multicores [5] and GPUs [35], and deployed in industrial domains [36]. In all these works, diversity is principally achieved by implementing SRE. However, SRE does not effectively protect the systems for CCFs that do not exhibit a transient behaviour (e.g., the ones originated in the register file due to wearout or systematic defects). As long as small technology nodes (e.g., 7nm) are deployed in safety-related areas, the introduction of new diversity techniques to cope with increasing fault rates becomes mandatory [13].

Several mechanisms to protect SRAM structures from wearout processes have been also proposed in the literature for CPUs [37] and GPUs [38]. These mechanisms target bias temperature instability (NBTI) wearout effects and thus, are orthogonal to our proposal. Likewise, work in [39] evinces that balancing RF utilization in MIPS processors alleviates the NBTI-related RF wearout. The impact of randomized cache designs in HCI-related wearout effects have also been proposed in [40]. Similarly as our mechanism does, randomized cache designs distribute cache accesses uniformily across the different cache sets, reducing the impact of the HCI-related degradation. Interestingly, randomized cache designs –although not evaluated yet– do also have the potential to improve the diversity of homogenous core designs.

VI. Conclusions

Mitigation of common-cause faults (CCFs) becomes an important concern in the design of resilient systems. As it has been experimentally shown in this paper, staggered redundant execution is rather inefficient for protecting multicore processors against CCFs. On the one hand, this is due to the very high criticality of core processor structures (such as register file), which, being affected by a fault, are very likely to lead a core (hardware thread) to failure under most runtime conditions. On the other hand, a very pronounced non-uniformity in utilization of RF registers leads to quicker wearout of the most utilized RF registers.

This paper has proposed a register file randomization mechanism (RFR) that addresses the two aforementioned problems. First, by providing a diverse mapping between logical and physical registers across processor cores, it makes that critical logical registers are less likely to be simultaneously affected by CCFs. Our experiments have shown that this improves the robustness of the resulting system up to an order of magnitude. Second, by periodically remapping logical registers to a different set of physical registers (by changing a random key), it also equalizes the utilization rate of physical registers. This reduces the electrical stress suffered by the most active physical registers, and extends the overall system lifetime (by 4 to 7 times in our presented case study).

Future work should study the possibility of using randomized register remapping directly at the level of processor pipeline, in order to further improve the robustness (and potentially the security) features of processor cores. Some particular problems to be addressed in this way are the special usage of some architectural registers and the implicit assumptions imposed by compilers on such registers.

VII. Acknowledgment

This work has received funding from the ECSEL Joint Undertaking (JU) under grant agreement No 877056 and the Agencia Estatal de Investigación from Spain under grant agreement no. PCI2020-112092, and from the the European Unions Horizon 2020 research and innovation programme under grant agreement no. 871467.

REFERENCES

[1] L. Liu, S. Lu, R. Zhong, B. Wu, Y. Yao, Q. Zhang, and W. Shi, "Computing systems for autonomous driving: State of the art and challenges," *IEEE Internet of Things Journal*, vol. 8, no. 8, pp. 6469–6486, 2021.

[2] STMicroelectronics, "32-bit Power Architecture microcontroller for automotive SIL3/ASILD chassis and safety applications," 2014.

[3] Infineon, "AURIX Multicore 32-bit Microcontroller Family to Meet Safety and Powertrain Requirements of Upcoming Vehicle Generations," http://www.infineon.com/cms/en/about-infineon/press/press-releases/2012/INFATV201205-040.html.

[4] P. Okech, N. M. Guire, and W. Okelo-Odongo, "Inherent diversity in replicated architectures," 2015.

[5] S. Alcaide et al., "Software-only diverse redundancy on GPUs for autonomous driving platforms," in *IOLTS*, 2019.

[6] International Standards Organization, *ISO/DIS 26262. Road Vehicles – Functional Safety*, 2009.

[7] International Electrotechnical Commission , *Functional Safety of Electrical/Electronic/Programmable Electronic Safety-related Systems (E/E/PE, or E/E/PES) (Edition 2.0)*, 2010.

[8] D. Shapiro, "Introducing Xavier, the NVIDIA AI Supercomputer for the Future of Autonomous Transportation," *NVIDIA blog*, 2016. [Online]. Available: https://blogs.nvidia.com/blog/2016/09/28/xavier/

[9] A. Platschek, N. Guire, and L. Bulwahn, "Certifying linux: Lessons learned in three years of sil2linuxmp," 02 2018.

[10] M. Fockel, "Safety requirements engineering for early sil tailoring," Dissertation, Fakultät für Elektrotechnik, Informatik und Mathematik, Universität Paderborn, Dec. 2018.

[11] X. Iturbe, B. Venu, J. Jagst, E. Ozer, P. Harrod, C. Turner, and J. Penton, "Addressing Functional Safety Challenges in Autonomous Vehicles with the Arm Triple Core Lock-Step (TCLS) Architecture," *IEEE Design and Test*, vol. PP, no. 99, pp. 1–1, 2018.

[12] G. C. Medeiros, M. Fieback, L. Wu, M. Taouil, L. M. B. Poehls, and S. Hamdioui, "Hard-to-detect fault analysis in finfet srams," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, pp. 1–14, 2021.

[13] L. Condra, A. Alagappan, and C. Hillman, "Sae arp6338: Process for assessment and mitigation of aging and potential early wearout of life-limited microcircuits (llm)," *SAE International Journal of Advances and Current Practices in Mobility*, vol. 1, no. 4, pp. 1653–1660, apr 2019. [Online]. Available: https://doi.org/10.4271/2019-01-1254

[14] Q. Zhang, A. Z. Mohammed, Z. Wan, J.-H. Cho, and T. J. Moore, "Diversity-by-design for dependable and secure cyber-physical systems: A survey," 2020.

[15] S. Mittal and R. Blanton, "Learnx: A hybrid deterministic-statistical defect diagnosis methodology," in *2019 IEEE European Test Symposium (ETS)*, 2019, pp. 1–6.

[16] M. T. Rahman, A. Hosey, Z. Guo, J. Carroll, D. Forte, and M. Tehranipoor, "Systematic correlation and cell neighborhood analysis of sram puf for robust and unique key generation," *Journal of Hardware and Systems Security*, vol. 1, 06 2017.

[17] P. Tummeltshammer, "Analysis of common cause faults in dual core architectures," PhD dissertation, Technische Universitat Wien, 2009.

[18] H. Amrouch and J. Henkel, "Self-immunity technique to improve register file integrity against soft errors," in *2011 24th Internatioal Conference on VLSI Design*. IEEE, 2011, pp. 189–194.

[19] ARM, "Arm trustzone true random number generator: Technical reference manual." 2017, http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.100976_0000_00_en.

[20] P. A. et. al, "Efficient Shift Registers, LFSR Counters, and Long Pseudo-Random Sequence Generators, Xilinx application note XAPP052." https://www.xilinx.com/support/documentation/application_notes/xapp052.pdf.

[21] A. González, M. Valero, N. Topham, and J. M. Parcerisa, "Eliminating cache conflict misses through xor-based placement functions," in *Proceedings of the 11th International Conference on Supercomputing*, ser. ICS '97. New York, NY, USA: Association for Computing Machinery, 1997, p. 76–83. [Online]. Available: https://doi.org/10.1145/263580.263599

[22] S. McFarling, "Combining branch predictors," Citeseer, Tech. Rep., 1993.

[23] R. Peñaranda, C. G. Requena, M. E. Gómez, and P. López, "Xor-based hol-blocking reduction routing mechanisms for direct networks," *Parallel Comput.*, vol. 67, pp. 57–74, 2017. [Online]. Available: https://doi.org/10.1016/j.parco.2017.06.004

[24] H. Kim, S. B. K. Boga, A. Vitkovskiy, S. Hadjitheophanous, P. V. Gratz, V. Soteriou, and M. K. Michael, "Use it or lose it: Proactive, deterministic longevity in future chip multiprocessors," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 20, no. 4, Sep. 2015. [Online]. Available: https://doi.org/10.1145/2770873

[25] R. Zhang, T. Liu, K. Yang, and L. Milor, "Analysis of time-dependent dielectric breakdown induced aging of sram cache with different configurations," *Microelectronics Reliability*, vol. 76-77, pp. 87–91, 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0026271417302287

[26] C. Gaisler, *NOEL-V Processor*, 2020, https://www.gaisler.com/index.php/products/processors/noel-v.

[27] A. Waterman, Y. Lee, D. A. Patterson, and K. Asanović, "The risc-v instruction set manual, volume i: User-level isa, version 2.0," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2014-54, May 2014. [Online]. Available: http://www2.eecs.berkeley.edu/Pubs/TechRpts/2014/EECS-2014-54.html

[28] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown, "MiBench: A free, commercially representative embedded benchmark suite," in *IEEE 4th Annual Workshop on Workload Characterization*, 2001, pp. 3–14.

[29] J. Gustafsson, A. Betts, A. Ermedahl, and B. Lisper, "The mälardalen wcet benchmarks: Past, present and future," in *10th International Workshop on Worst-Case Execution Time Analysis (WCET 2010)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2010.

[30] "Tiny aes in c," https://github.com/kokke/tiny-AES-c.

[31] I. Tuzov, D. de Andrés, and J.-C. Ruiz, "Davos: Eda toolkit for dependability assessment, verification, optimisation and selection of hardware models," in *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 2018, pp. 322–329. [Online]. Available: https://doi.org/10.1109/DSN.2018.00042

[32] Xilinx Inc., "Vivado Design Suite User Guide. Synthesis. UG901 (v2017.4)," 2017. [Online]. Available: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2017_4/ug901-vivado-synthesis.pdf

[33] C. McNairy and R. Bhatia, "Montecito: a dual-core, dual-thread itanium processor," *IEEE Micro*, vol. 25, no. 2, pp. 10–20, 2005.

[34] J. Andersson, J. Gaisler, and R. Weigand, "Next generation multipurpose microprocessor," *DASIA*, 08 2010.

[35] S. Alcaide Portet, L. Kosmidis, C. Hernandez, and J. Abella, "Software-only triple diverse redundancy on gpus for autonomous driving platforms," in *2020 50th Annual IEEE-IFIP International Conference on Dependable Systems and Networks-Supplemental Volume (DSN-S)*, 2020, pp. 82–88.

[36] A. Gerstinger, H. Kantz, and C. Scherrer, "Tas control platform: A platform for safety-critical railway applications," *ERCIM News*, vol. 2008, 2008.

[37] J. Abella, X. Vera, and A. Gonzalez, "Penelope: The nbti-aware processor," in *40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 2007)*, 2007, pp. 85–96.

[38] A. Valero, F. Candel, D. Suárez-Gracia, S. Petit, and J. Sahuquillo, "An aging-aware gpu register file design based on data redundancy," *IEEE Transactions on Computers*, vol. 68, no. 1, pp. 4–20, 2019.

[39] H. Amrouch, T. Ebi, and J. Henkel, "Stress balancing to mitigate nbti effects in register files," in *2013 43rd Annual IEEE/IFIP international conference on Dependable Systems and Networks (DSN)*. IEEE, 2013, pp. 1–10.

[40] D. Trilla, C. Hernandez, J. Abella, and F. J. Cazorla, "Aging assessment and design enhancement of randomized cache memories," *IEEE Transactions on Device and Materials Reliability*, vol. 17, no. 1, pp. 32–41, 2017.