CEVERO: A soft-error hardened SoC for aerospace applications

Igor Macedo Silva*, Otávio do Espírito Santo*, Diego V. Cirilo do Nascimento† and Samuel Xavier-de-Souza*
*Centro de Tecnologia
Universidade Federal do Rio Grande do Norte, Natal, Brazil
Email: igormacedo@ufrn.edu.br, otavio10espsanto@gmail.com, samuel@dca.ufrn.br
†Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Norte, Natal, Brazil
Email: diego.cirilo@ifrn.edu.br

***Abstract*—Radiation in the space environment poses a constant threat to circuit devices, and, as a solution, several fault tolerance techniques have been developed. However, some techniques use either proprietary technology or radiation-hardening specialized foundries to manufacture, either of which are not easily accessible to most researchers. There is, yet, another set of rad-hard techniques which provide sufficient fault tolerance to specific aerospace applications. The CEVERO SoC is a custom proof-of-concept design that implements techniques as lockstep, dual module redundancy and state recovery in its inner fault tolerance module. It is meant to integrate with the open-source PULP platform and to open a path to a full fault-tolerant SoC.**

***Index Terms*—Radiation-Hardened, Fault tolerance, Soft error, lockstep, DMR, CEVERO.**

I. Introduction

Since its beginnings, space exploration was a driving force towards technological advancement. The microelectronics industry has been a key factor in the human endeavor to discover and study the cosmos and its celestial bodies. However, as expected from any pioneer venture, space environment has confronted our electronic appliances with unforeseen problems and, in order to tackle them, there was need to develop highly specialized technology.

One such problem posed onto electronic devices in space is radiation-induced fault. This effect is caused when highly energetic particles strike a sensitive region of a microelectronic circuit. In some cases, the strike may cause no noticeable effect, but in other cases it might cause a transient disruption of a circuit operation, a logic state change or a permanent defect in the microelectronic circuit [1].

Several radiation hardening techniques have been implemented since, so as to avoid errors from happening or to allow electronic devices to recover from them. These techniques are present in several circuitry abstraction levels, however, low-level techniques used in the foundry process or gate design usually require technology that is either inaccessible to most (such as rad-hard processes in specialized foundries) or out-dated for current performance requirements. As an alternative, researchers have shifted their efforts to higher levels of abstraction, specially the architectural level.

Current soft error tolerance chips implement one or several of these architectural protection circuits inside the general processing unit or they even use Custom Out of The Shelf (COTS) chips to create the fault tolerant device. Following this line of thought, this document proposes the usage of an existing, growing, high performance and ultra low power platform called Parallel Ultra Low Power (PULP) platform, based on the RISC-V instruction set, to implement a fault tolerant system concept which requires minimal modifications to the current processor design.

The next section will provide the reader with a brief introduction on radiation induced soft error causes and the techniques used to mitigate such errors. It presents an example of processor that implements error tolerance on the architectural level and exemplifies the reason these techniques are indeed effective. Following, section III describes the PULP-based system, focusing on the fault tolerance module description and how it interfaces with processor units and memory. Section IV exposes the methodology and its results to validate the proposed system as fault tolerant and the last section concludes with an overview of the proposed system and future developments intended to this project.

II. Radiation-induced Errors and Mitigation Techniques

Some of the first radiation effects ever described in electronics date back from the 1950s [2]. However it was only in the 1970s and 1980s that this phenomena caught the attention of the scientific community and was then extensively studied [3]–[5].

The reason radioactive particles induce faults in electronic devices is due to a disturbance in the electric field, which generates electron-hole pairs in sensitive parts of a transistor. These charged particles are then free to be collected by diffusion regions and, later, reach device contacts, producing incorrect signals.

A. Radiation-Hardening Techniques

The understanding of how highly energetic particles disrupt the correct work of components in a microelectronic circuit lead to the development of techniques to avoid incorrect behavior. Theses techniques are applied from the lowest abstraction level up until the highest one, and it is a common procedure to apply as many radiation-hardening techniques as necessary

to guarantee that the circuit will be hardened and tolerant to a diverse range of faults caused by radiation.

Radiation-Hardened by Process (RHBP) is the first class of techniques to shield microelectronic circuits against energetic particles. These techniques focus on enhancing the process on which transistors are created by using selected materials, insulation layers, doping levels, or using proprietary fabrication steps. They usually need dedicated foundry lines to be fully developed [6]. The next group of techniques with an increasing abstraction level are Radiation-Hardened by Design (RHBD) techniques. Generally, approaches that fall into this category deal with circuit layout and composition of logic gates and memory cells. The most compelling argument in favor of this approach is the usage of commercially available processes instead of rad-hard processes from specialized foundries.

Lastly, Radiation-Hardened by Architecture (RHBA) techniques are generally implemented inside modules in the processor micro-architecture such as register files or arithmetic logic units, adding extra fault-tolerance features to them, as explained below. An advantage in favor of RHBA techniques is the fact they may be implemented with very high level tools, such as hardware description languages (HDLs) and electronic design automation software, when compared to previously mentioned techniques, and they still take advantage of hardware execution speeds. The most common architectural features that allow fault tolerance are N-Modular Redundancy (N-MR), data scrubbing and Error Correction Codes (ECC).

Generically put, N-MR consists of replicating a digital component and comparing the output of these components. In case of an error, the output of the components will differ and the RHBA technique will detect the error. The most common applications of N-MR are Dual Modular Redundancy (DMR) and Triple Modular Redundancy (TMR), and they differ by how resilient they are against error. In the case of the DMR, the only way to avoid error propagation would be to halt whatever computation was performed and re-execute it (note that, in some cases, the previous circuit state has to be saved and restored before re-executing). However, in the case of TMR (and higher redundancy levels as well), the comparison circuit, besides detecting the error, can infer which is the most likely correct result by a voting process, and then forward this result to the module output. In each of these methods, there is a parameter penalty that should be considered: although TMR might be quicker in acquiring the correct output because there is no computation re-execution, the DMR is probably the most area-optimized solution because the digital component is replicated only twice as many as the original one (against three times as many when considering the TMR solution) and the error-detection hardware should be simpler than the voter circuit required for TMR.

Another very common technique to detect and correct errors is memory scrubbing, typically associated with Error Correction Codes (ECC) [7], [8]. Because, soft-errors might occur in memory regions that are not actively used, many of those errors can accumulate if left unchecked and, in a worst case scenario, they might become incorrigible errors because the ECC will not be able to define which bits are wrong. So, in order to avoid this, memory components should implement a data scrubbing technique which continuously browses memory positions and compares their values against the ECC. In case an error is detected, the hardware "scrubber" will use ECC to correct and write the expected value at the memory position it is due. An important consideration is that the scrubber should operate at a defined frequency in order to avoid the accumulation of errors [7].

B. Fault-Tolerant Processors

The research on all these Radiation-Hardened techniques culminates in the development of digital circuits for space-borne applications. The heart of most computations systems is a general purpose processor that receives the binary code and executes the instructions of the specified Instruction Set Architecture (ISA). Therefore, in order to deploy computational systems for usage in natural space environment, engineers have developed a diverse range of radiation-hardened processors that implement several of the strategies discussed before.

However, as technologies advance, many of the mission-proven processors start to become obsolete. Performances like the RAD750 [9] which achieves 200MHz and the RH3000 [10], 25MHz, do not seem to be sufficient for future space applications and they are, for sure, several generations behind the current commercially available processors. In fact, several of the modern processors for space applications tend to focus on RHBA techniques, presumably because their ease of development, verification and the fact that theses techniques allow to use the full power of today's commercially available transistor processes. One interesting example is the Arm Triple Core Lock-Step (TCLS) processor.

The Arm TCLS implements a TMR approach, in which three cores execute in lock-step. When a divergence is detected between selected signals in each core, the system enters a correction state that defines the correct result by a voting process and restores the correct state into the cores.

The chip is based on the ARM Cortex-R5 and is deployed in applications that require high dependability, predictability and availability in safety-critical systems such as in business enterprise or automotive industry. It bets on the usage of Commercially-off-the-shelf processors (COTS), relying on minimal use of RHBD technology (amounting to 4% of the solution) to cope with the radiation effects of radiation-induced faults. The authors claim that in Low Earth Orbits (LEO), the system could work with no RHBD elements, relying exclusively with the RHBA techniques to deal with sporadic soft errors in its critical parts [11].

III. A SOFT-ERROR HARDENED SOC PROPOSAL

Several fault tolerant processors and chips are based on existing designs. This is the course that the Arm TCLS chose to take. This design uses processor that already exists and improves over that processor with different and customized solutions for fault tolerance.

The CEVERO (Chip multi-procEssor for Very Energy efficient aeRospace missiOns) SoC takes this idea to a platform called PULP, which is based on the open-source and fast-growing instruction set RISC-V. As a platform focused on low energy consumption, high processing capacities and wide range of applications, the PULP platform [12], [13] is an extremely good and open-source starting point to develop a fault-tolerant processor SoC.

As a proof of concept, the CEVERO SoC is described here in a simplified SoC organization. The goal is to describe the inner workings of the fault tolerant module as developed and how it is integrated to processing cores from the PULP platform.

A. CEVERO SoC organization

The system is composed by an instruction memory, data memory, two ibex cores (formerly known as Zero-riscy [14]) and a fault tolerance module that communicates with both cores. Together they form the CEVERO SoC. This SoC uses the duplex architecture technique, where the general processing units are duplicated.

The adopted approach to achieve fault tolerance in this system is to verify that both cores execute the same instruction and compare their results at the time of writing to the register file, which is the last execution step of an instruction in the core pipeline. Using this strategy, the system is able to verify instruction by instruction if an error occurred. This technique where the system runs the same set of operations at the same time in parallel is denominated lockstep, and it fits very well to a in-order processor like the ibex core.

In this deterministic processor, it is possible to save the state of a process and later resume the process by restoring the internal state. Similar techniques are used by operating systems during context switching and by commercial processors for simultaneous multithreading. Comparably, this fault-tolerance module saves the state of the process that is running at that moment. Thus, when an error occurs, it is possible to recover the system to a safe state.

The Figure 1 shows the internal organization of the SoC and the interconnections between the internal components. The FTM block represents the fault tolerance module (FTM), which communicates with both ibex cores. Also, the ibex cores share the same instruction bus and data memory bus, since they execute completely in parallel and should read and write the same data from and to both memories. Core 0 is assigned the function to fetch the instruction in the memory and to write in data memory. The core 1 will get the same instruction as core 0 by pulling this information from the bus between core 0 and memory.

The data flow in this SoC architecture is similar to a conventional processor, except for the writing stage of the pipeline. The program is stored in the instruction memory. Then, both cores fetch the same instructions from memory. After processing, the result is stored in the General Purpose Register file (GPR). At this moment, the FTM is responsible for intercepting the result and also saving it to the Safe General Purpose Register (SGPR), which should be a register file with embedded error correction, hardened against soft errors from radioactive particles.

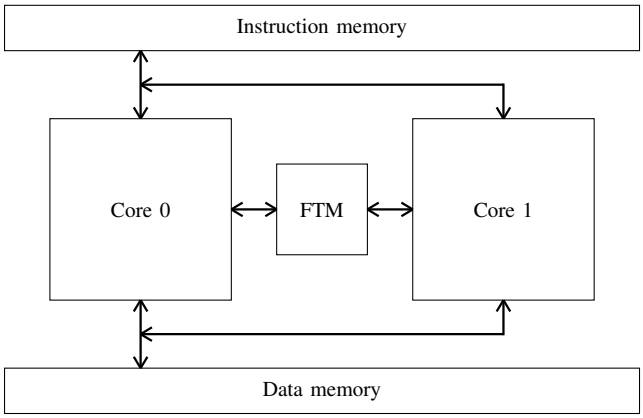

Fig. 1. CEVERO SoC organization.

B. Ibex core modification

The ibex core has undergone minor modifications to suit this project. One signal and two buses were exposed out of the core, meaning they also became I/O ports. The signal refers to `write_enable` signal, the other two buses refer to `address` and `write_data`. All these are destined to the internal GPR.

The `write_enable` signal is used to allow the write operation in the GPR. The `address` is used for selecting the address where the data will be written and the `data` signal is the data itself. These same signals and buses are driven to the SGPR, located within the fault tolerance module.

C. Fault tolerance module

The Fault Tolerance Module (FTM) is responsible for detecting and correcting any error that occurs in the execution of the program. When this block detects an error, it halts both cores and returns the execution point to a safe state. The module then resumes execution from this safe state and guarantees the correct execution of the program.

The FTM receives the `write_enable` signal, the address and the data buses from both cores. Those signals are internal to the ibex core and they are intended to the GPR of both ibex cores. With this data, the FTM module is able to compare results and detect any mismatches.

Interconnections among components that make up the module are shown in Figure 2: the Controller, the Comparator, the Safe General Purpose Register (SGPR) and the Safe Program Counter (SPC). The components that make up the module will be discussed.

1) Comparator: The comparator receives signals and buses from both cores and performs a bitwise comparison in each one of them. If all the bits from one core are equal to the bits in the other, the address and the data are routed to the SGPR and the execution in the cores proceeds normally. Otherwise,

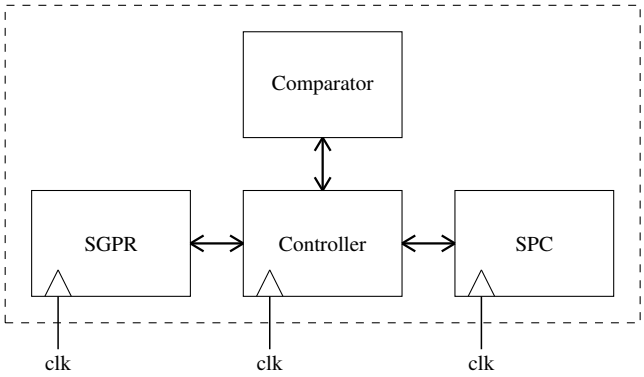

Fig. 2. Internal components of the module.

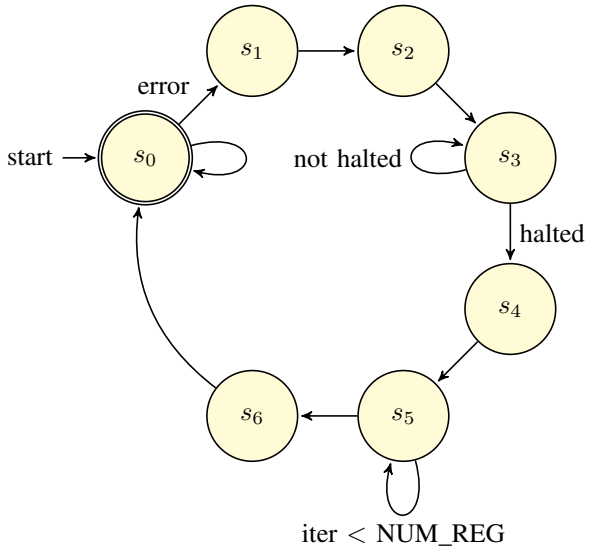

Fig. 3. Controller state machine.

if one bit is different, the comparator will send an error signal to the controller.

In order to perform this comparison, all information which originates from the same core is concatenated into a single bit stream. Then a bitwise XOR logic operation is executed between both sequences from each core. In a XOR operation, different bits evaluate to true and equal bits, to false. Therefore, whenever there is a bit mismatch, the error signal is generated from this operation.

2) SGPR: The SGPR is a register file identical to the GPR existent in the ibex core. This block receives the address and data from the comparator if this information is equal in both the cores.

When an error is detected, the write enable signal changes to false. At this moment, no more data is written to the block and the recovery process starts. The controller begins to access each address of the SGPR so that the information contained in each of them is sent to the GPR in both cores in order to restore the previous safe state.

3) SPC: the SPC receives and stores the program counter value. It continuously stores the PC value at each fetch done by the core. The value stored is always the one in the PC from two cycles ago, because of the existence of a two stage pipeline.

When an error is detected, the controller will receive the value contained in this block and it will overwrite the program counter of the ibex cores.

4) Controller: The controller block is responsible for managing the entire module, controlling the SGPR, SPC and the ibex cores during the recovery process.

This component receives two signals: the error signal from the comparator and the halted signal originated from the core. The first signal is high when the comparator detects a difference between values coming from both cores. The other signal is high when the core is in halted state.

Figure 3 shows the controller state machine. It is composed of 7 states. The first state $S0$ is the wait mode where the FTM waits until an error occurs. In $S0$, all internal signals and variables are restarted. When an error happens, the controller starts the correction cycle. In the next state $S1$, the controller sends a `reset` signal to both cores, clearing all memory elements. Then writing to the SPC is disabled. In $S2$, `halt` signal is sent, the machine goes to $S3$, and the controller waits until the cores respond with the `halted` signal. All these steps represent the communication protocol between the FTM and the cores. The protocol is described in more detail in subsection III-E.

In state $S4$, the controller sends the value of the SPC to the PC in both cores. The $S5$ state is where the controller addresses all registers of the GPRs and SGPR fetching the value contained in each register of the SGPR to overwrite the value in the registers of the GPR with corresponding address. When this process is finished, i.e. when the controller is done accessing all registers, the operation is terminated and then the block advances to the next state. In the final state, the `resume` signal is sent and the ibex cores leave the halted state.

D. Debug interface communication

The fault tolerance module uses the debug interface of the ibex core to establish the connection with it. The choice to use this interface rather than creating a totally new one is mainly due to the fact that there is no need to modify the internal structure of the core. Creating a new interface would considerably change the core unit and would be a very intrusive approach.

The focus of this stage of the project is to develop a fault tolerance module able to detect and correct an error that occurs when the SoC is in operation. It is not in best interest to develop a processor or modify one profoundly. That being said, using the debug interface is convenient and strictly needed to the recovery process. In future developments, this debug interface shall be adapted and integrated into production hardware in order to allow state recovery.

Using the debug interface allows, with some ease, to control the core and change values of the internal memory elements. This technique enables the system to pause the core, so it will

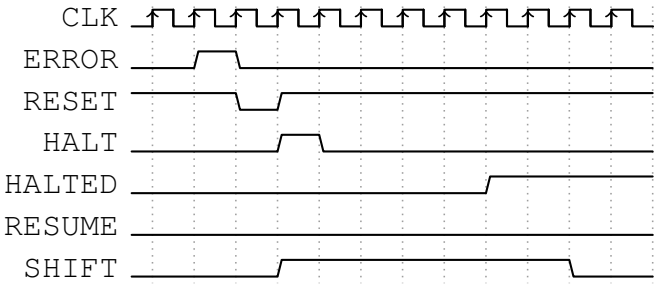

Fig. 4. Beginning of the correction operation.

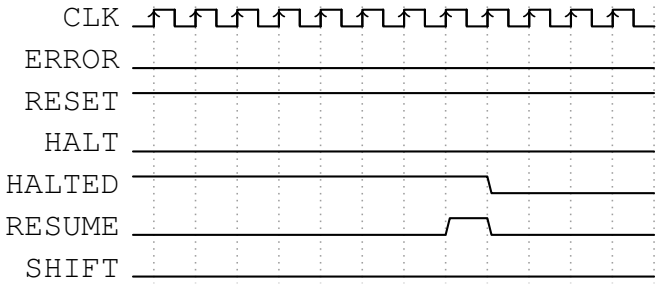

Fig. 5. End of the correction operation.

not fetch new instructions while the FTM controller performs the state recovery.

The memory elements are accessible through a debug address port. The address bus of the debug interface has 15 bits. This bus is considerably broad, because it accesses multiple distinct registers, such as the GPR registers, the PC, and the CSR registers, for example. In addition, the address between memory blocks are well spaced apart.

The address to access the GPR varies from **0x400** to **0x47F**, which corresponds to the range that covers the address of 32 registers. The memory is byte addressed, then every 4 bytes corresponds to a register. The PC is accessed by address **0x2000**. The information inserted in this location corresponds to the value of the next PC, which will be used to fetch the next word in the instruction memory. All address are accessible only if the core is halted.

E. Proposed protocol

In this section, the communication protocol between the fault tolerance module and the ibex core is explained.

Figure 4 shows the digital timing diagram of the proposed protocol. The error signal is an internal signal. Here it has been made explicit to expose the moment when the error detection occurs. This signal originates from the comparator block.

As shwon in figure 4, the FTM detects an error signal coming from the comparator, which happens when there is a difference between bit sequences fetched from both cores. The controller then sends a reset signal to the ibex cores, deleting all stored memory elements. Soon after, in the next clock cycle, the `halt` and `shift` signals are driven high and the halt signal is kept high for one clock cycle. There is a delay of four clock cycles between the FTM controller sends the `halt` signal and the core answers that it has been `halted`. After the core responds with the `halted` signal, the `shift` lasts for two clock cycles.

There is only one bus to address all the register through the debug interface. The same goes for reading and writing data. For this reason, multiplexers are used in the design of the project to switch between memory elements. The `shift` signal is sent to the multiplexers for this transition to happen.

At the end of the correction process, the `resume` signal is sent. Then, both cores leave halted mode, the `halted` signal changes to low logic level, and then the cores continue with normal processing. But now resuming from a safe state before the error had occurred. Figure 5 shows the active `resume` signal and the cores' exit from halted mode.

IV. System Validation

A. Methodology

The approach to test the system is relatively simple and intended to validate basic tolerance to bit flips in the system. As a project in its initial phase, this methodology suits its purpose to validate system error recovery and to illustrate the system's feasibility in the PULP platform.

The testing procedure is described as follows: a program is loaded into instruction memory and execution begins. Both cores should execute the same instructions at the same time and, at any given time, an error is induced by a built-in fault injection module in the program instruction. It changes the instruction of one of the cores, forcing a different result from the other. This intervention forces the fault tolerance module to go into correction mode operation and to correct the error by returning execution of the program to a safe state. There is no difference on where the error occurs in the ibex core because, by the end of the pipeline, if any of the values from both cores differ, the recovery process is the same and should take the same amount of time. However, a permanent fault would cause this recovery process to carry on indefinitely, rendering this system ineffectual.

The main program used for testing was the code written in assembly that calculates up to the n-th element of the Fibonacci sequence. The assembly code was written using Ripes [15], a simulator and assembly code editor built for RISC-V. With this program, the binary equivalent was created to be embedded into instruction memory.

B. Validation

One of the applied tests was to replace an entire instruction word during the execution process in one of the cores. This provoked a change of several bits compared to the instruction word that would be executed if there was no intervention.

The fault injection module was implemented between the instruction memory and the core. When it receives an error signal, it shifts its output to a faulty instruction. The `error` signal is defined in the project's testbench. When this signal is sent, the core will fetch the wrong instruction and, therefore,

an error is forced because the both cores will have different instructions to process at the same instant.

V. Results

The tested program took 82 clock cycles to run, outputting the value 89 which is the expected twelfth term of the Fibonacci sequence. At a certain point in the execution of the program, an error signal was introduced so that core 0 fetches a wrong instruction, causing an error in execution. This test was executed multiples times differing only by when the fault is injected.

The module proved effective in detecting the error at the exact moment when a difference between program instructions coming from either one of the cores.

The FTM took a total of 40 clock cycles to complete the entire process. Including the time from error detection up until error correction.

VI. Conclusion

The natural space environment is conclusively very hazardous for digital circuits as radiation related problems occur frequently with circuitry in spaceborne applications. However, a big variety of radiation-hardened techniques were developed to mitigate and even extinguish the effect of those problems in digital components of a computer.

Some of these techniques, specially RHBP and RHBD ones, require proprietary and/or really costly technologies, which renders them inaccessible to many researchers. Nevertheless, other techniques with higher abstraction levels are more accessible and suitable for specific space applications.

This document proposes the implementation of a few of these techniques, such as lock-stepped DMR in conjunction with state recovery and ECC protected register file in a fault tolerance module integrated to an open-source core from the PULP platform. It realizes a fault-tolerant system, validated by a test program, that is replicable due to its open-source foundation and readily available technologies. As explained in past sections, the FTM itself and its integration to the ibex core need modifications so as to become a manufacturable system: the debug interface shall be adapted and condensed down to allow strict access to register files, and the FTM should consider implementing scrubbing and other fault tolerance techniques in its inner components to provide full coverage against radiation-induced soft errors.

As it is, this proof-of-concept system opens a path to the future development of a more robust system-on-a-chip. This SoC will be based on a custom PULP SoC, which shall be deliberately protected with similar radiation hardening techniques, and will include this proposed fault tolerance module. Therefore, there is an increasingly complex, yet viable path to future works on this system, and the outcome is a system-on-a-chip which advocates on a more accessible, open-hardware spaceborne system.

References

[1] P. E. Dodd and L. W. Massengill, "Basic mechanisms and modeling of single-event upset in digital microelectronics," *IEEE Transactions on Nuclear Science*, vol. 50 III, no. 3, pp. 583–602, 2003.

[2] J. F. Ziegler, "Terrestrial cosmic rays," *IBM Journal of Research and Development*, vol. 40, no. 1, pp. 19–39, 1996.

[3] D. Binder, E. C. Smith, and A. B. Holman, "Satellite Anomalies from Galactic Cosmic Rays," *IEEE Transactions on Nuclear Science*, vol. 22, no. 6, pp. 2675–2680, 1975. [Online]. Available: http://ieeexplore.ieee.org/document/4328188/

[4] J. C. Pickel and J. T. Blandford, "Cosmic ray induced errors in MOS memory cells," *IEEE Transactions on Nuclear Science*, vol. 25, no. 6, pp. 1166–1171, 1978.

[5] C. S. Guenzer, E. A. Wolicki, and R. G. Alias, "Single event upset of dynamic rams by neutrons and protons," *IEEE Transactions on Nuclear Science*, vol. 26, no. 6, pp. 5048–5052, 1979.

[6] A. Keys, J. Adams, D. Frazier, M. Patrick, M. Watson, M. Johnson, J. Cressler, and E. Kolawa, "Developments in Radiation-Hardened Electronics Applicable to the Vision for Space Exploration," in *AIAA SPACE 2007 Conference & Exposition*. Reston, Virigina: American Institute of Aeronautics and Astronautics, sep 2007, pp. 1–10. [Online]. Available: http://arc.aiaa.org/doi/10.2514/6.2007-6269

[7] S. S. Mukherjee, J. Emer, T. Fossum, and S. K. Reinhardt, "Cache scrubbing in microprocessors: Myth or necessity?" *Proceedings - IEEE Pacific Rim International Symposium on Dependable Computing*, pp. 37–42, 2004.

[8] B. Schroeder, E. Pinheiro, and W. D. Weber, "DRAM errors in the wild: A large-scale field study," *Communications of the ACM*, vol. 54, no. 2, pp. 100–107, 2011.

[9] R. W. Berger, D. Bayles, R. Brown, S. Doyle, A. Kazemzadeh, K. Knowles, D. Moser, J. Rodgers, B. Saari, and D. Stanley, "The RAD750™ - A radiation hardened PowerPc™ processor for high performance spaceborne applications," in *IEEE Aerospace Conference Proceedings*, vol. 5. IEEE, 2001, pp. 52 263–52 272. [Online]. Available: http://ieeexplore.ieee.org/document/931184/

[10] M. Iacoponi, "The RH-3000 MIPS compatible space processor," in *9th Computing in Aerospace Conference*. Reston, Virigina: American Institute of Aeronautics and Astronautics, oct 1993, pp. 1–6. [Online]. Available: http://arc.aiaa.org/doi/10.2514/6.1993-4469

[11] X. Iturbe, B. Venu, E. Ozer, J. L. Poupat, G. Gimenez, and H. U. Zurek, "The Arm triple core lock-step (TCLS) processor," *ACM Transactions on Computer Systems*, vol. 36, no. 3, 2019.

[12] D. Rossi, F. Conti, A. Marongiu, A. Pullini, I. Loi, M. Gautschi, G. Tagliavini, A. Capotondi, P. Flatresse, and L. Benini, "PULP: A parallel ultra low power platform for next generation IoT applications," *2015 IEEE Hot Chips 27 Symposium, HCS 2015*, 2016.

[13] D. Rossi, A. Pullini, I. Loi, M. Gautschi, F. K. Gurkaynak, A. Teman, J. Constantin, A. Burg, I. Miro-Panades, E. Beigne, F. Clermidy, P. Flatresse, and L. Benini, "Energy-Efficient Near-Threshold Parallel Computing: The PULPv2 Cluster," *IEEE Micro*, vol. 37, no. 5, pp. 20–31, 2017.

[14] P. D. Schiavone, F. Conti, D. Rossi, M. Gautschi, A. Pullini, E. Flamand, and L. Benini, "Slow and steady wins the race? A comparison of ultra-low-power RISC-V cores for internet-of-things applications," *2017 27th International Symposium on Power and Timing Modeling, Optimization and Simulation, PATMOS 2017*, vol. 2017-Janua, pp. 1–8, 2017.

[15] "Ripes," 2018. [Online]. Available: https://github.com/mortbopet/Ripes