HW/SW Co-Design of a Reliable Deep Space System exploiting Application-profiled RAM Scrubbing

HW/SW Co-Design of a Reliable Deep Space System exploiting Application-profiled RAM Scrubbing

*Nicola di Gruttola Giardino, †Paolo Bernardi, *Sabrina Corpino, *Fabrizio Stesina
*Dept. of Mechanical and Aerospace Engineering, Politecnico di Torino, Torino
†Dept. of Computer and Control Engineering, Politecnico di Torino, Torino
name.surname at polito.it

***Abstract*—In recent years, the increasing interest in space exploration has brought a significant change in the design of avionic systems. The growing complexity required a shift from using only low-performance, space-grade components to more performant Commercial Off-The-Shelf (COTS) components. These systems are, however, more susceptible to radiation-induced effects, and as such, require the engineers to implement advanced techniques of hardening-by-design or by software to increase the tolerance to Single Event Upset induced faults. Traditional scrubbing mechanisms for RAM with ECC encoding operate under the assumption that the entire memory space must be continuously checked for errors. However, this approach proves inefficient for complex space missions with varying mission scenarios, as it unnecessarily consumes resources by checking unused memory regions while potentially allowing errors to accumulate in critical areas. The need for more efficient error mitigation strategies becomes particularly crucial as space missions become more sophisticated and resource-constrained. This work proposes an innovative memory profiling-based scrubbing mechanism leveraging Hardware-Software Co-Design principles. Our approach introduces a specialized scrubber designed around the memory system, which interfaces with a register file accessible as a memory-mapped peripheral. The methodology involves a three-stage process: initial application compilation, memory profiling to determine occupation patterns, and application modification to program the register file with profiling results. This targeted approach ensures that only actively used memory regions are scrubbed, significantly improving efficiency while maintaining robust error protection in critical memory areas. The proposed solution offers a practical balance between system reliability and resource utilization, particularly valuable for modern space missions where optimizing both performance and radiation tolerance is essential.**

***Index Terms*—FPGA, transient fault, SEU, space applications, ECC**

I. Introduction

In recent years, space exploration has seen remarkable advancements, particularly with the increasing complexity of deep-space and planetary surface missions such as planetary defense satellites [1], [2], Mars rovers, and lunar landers [3], [4]. These missions demand highly advanced systems capable of handling increasingly sophisticated tasks, from autonomous navigation over rough terrain to real-time data processing for scientific experiments.

As the push for smaller and more efficient spacecraft grows, miniaturization has become a key trend in space. This shift allows for the development of compact platforms that are essential for planetary exploration, where every gram of weight and millimeter of space counts. However, this miniaturization also presents challenges. Electronic components must endure extreme temperatures, dust and the effects of radiation, which can disrupt their operation.

To meet these challenges, many space missions are moving away from traditional space-grade components and opting for Commercial Off-The-Shelf (COTS) components. While COTS components are often more cost-effective and offer better performance, they come with their own set of issues, particularly regarding radiation tolerance. The unique environments of deep-space and planetary surfaces expose these components to various radiation sources, which can lead to Single Event Upsets (SEUs) and micro-Single Event Functional Interrupts (micro-SEFIs). These radiation-induced effects can cause temporary disruptions or permanent damage to the systems, making it crucial to find effective ways to protect against them. To enhance the reliability of COTS components, engineers typically employ two main strategies: hardware-based solutions that involve designing components to be more resistant to radiation and implementing error detection and correction techniques, and software-based methods that provide additional layers of fault tolerance. However, relying solely on traditional Error Correction Code (ECC) methods has its drawbacks. These methods often require continuous scanning of the entire memory, which can waste valuable processing time and resources, especially in complex missions where memory usage can vary significantly. Given these challenges, there is a pressing need for more efficient approaches to memory protection that can adapt to the dynamic nature of planetary missions. Such solutions must not only ensure reliability but also optimize resource usage, allowing for better performance in both surface and deep space exploration.

This paper proposes a novel memory profiling-based scrubbing mechanism that leverages a Hardware-Software Co-Design approach to address the limitations of traditional ECC scrubbing. The proposed methodology profiles memory usage

This work was carried out within the Space It Up! project funded by the Italian Space Agency (ASI) and the Italian Ministry of University and Research (MUR) under contract No. 2024-5-E.0, CUP No. I53D24000060005.

to identify active regions and focuses scrubbing efforts only on these areas, significantly reducing wasted clock cycles. By combining efficient resource utilization with robust error mitigation, this approach offers a practical solution for enhancing the reliability of COTS components in modern space applications, particularly for resource-constrained planetary exploration missions.

Experimental results demonstrate the effectiveness of this approach, revealing substantial improvements in scrubbing efficiency and resource utilization, thereby enhancing the reliability of COTS components in modern space applications, particularly for resource-constrained planetary exploration missions.

In Section II the paper provides some background about SEU induced transient faults and an overview of the state-of-the-art for BRAM-based SEU mitigation techniques. Section III-A explains the proposed hardware architecture, while Section III-B describes the memory profiling mechanism, and Section IV shows the experimental results obtained from the case study. Finally, Section V draws some conclusions.

II. Background

A. SEU Induced Transient Faults

The increasing miniaturization of electronic components [5], particularly in space applications, has led to greater susceptibility to radiation-induced effects. SEUs and micro-SEFIs have become critical concerns since their first identification in the late 1970s, particularly as spacecraft electronics have evolved to use more sophisticated and compact semiconductor technologies [6]. The space environment presents unique challenges due to various radiation sources, including Galactic Cosmic Rays (GCRs), Solar Energetic Particles (SEPs), and geomagnetically trapped particles, each contributing to potential system failures [7]. SEUs occur when a single ionizing particle strikes a sensitive region of a semiconductor device, causing a transient change in the state of a memory cell or logic circuit. While these changes are typically non-destructive, they can lead to data corruption or logical errors in system operation. Micro-SEFIs represent a more complex phenomenon, where a single event causes temporary functional interruptions affecting larger portions of the system, potentially leading to more severe operational impacts than simple bit flips.

The increasing reliance on COTS components in space applications, driven by cost and performance requirements, necessitates improved understanding and mitigation of SEU effects. This is particularly crucial for deep-space and planetary surface missions, where radiation exposure is more severe and traditional radiation-hardened components may be impractical due to cost or performance limitations. The need to balance reliability with system performance, while maintaining cost-effectiveness, drives current research in this field.

B. RAM-based SEU mitigation techniques

In modern radiation-intensive environments—particularly in space applications—the reliable operation of FPGAs is continually challenged by the effects of ionizing radiation, which can induce single event upsets (SEUs) in semiconductor memories. Block RAMs (BRAMs) [8], fundamental to user data storage in FPGAs, are especially vulnerable due to their dense structure and the miniaturization of device geometries. Early mitigation efforts primarily focused on safeguarding configuration memory through error-detecting and error-correcting codes as well as periodic scrubbing techniques. However, RAMs themselves exhibit a unique sensitivity profile that requires customized mitigation strategies. Investigations into RAM SEU behaviour have demonstrated that although built-in features such as single-error correction and double-error detection (SEC–DED) can correct isolated faults, more complex events, such as multi-cell upsets and micro-SEFIs), can overwhelm conventional error correction schemes. Fault injection experiments and radiation testing campaigns have underscored the limitations of approaches designed solely for configuration memory, highlighting the need for a mitigation system that specifically addresses RAM vulnerabilities. [9]

Based on these insights, current research has shifted towards integrating on-chip RAM sensors with adaptive mitigation mechanisms. Several prior works have combined scrubbing, ECC, and monitoring techniques to detect and correct SEUs without interrupting system operation. [10]–[12] In these systems, error counters and built-in fault detection logic are used not only to correct soft errors in real time, but also to estimate the instantaneous radiation-induced upset rate. This information, in turn, serves as a trigger for adaptive strategies such as dynamic modular redundancy or partial reconfiguration, which can be activated on demand when upset rates exceed predefined thresholds. Although customized RAM wrappers and specialized sensor architectures have successfully demonstrated low area and power overhead in laboratory settings and even in on-orbit prototypes, the challenge remains to achieve highly robust RAM SEU mitigation that addresses transient faults in a resource-efficient manner.

III. Proposed Methodology

Utilizing a scrubbing mechanism that continuously scans the entire memory can prove to be inefficient in complex space missions with dynamically changing scenarios. In such instances, multiple memory usage patterns may exist within a single mission. Traditional scrubbing techniques for RAM equipped with ECC encoding operate under the premise that the whole RAM must be routinely checked for errors. This method, however, incurs the cost of verifying memory regions that are never used by the system, resulting in wasted processing cycles while errors can build up in other critical areas. In contrast, the proposed method employs a hybrid hardware-software strategy that focuses solely on the active memory regions.

The proposed methodology, shown in Figure 1, is based on a scrubber, designed around the memory, which exploits a register file, accessible as a peripheral memory mapped in the system. With this approach, the application to be run is compiled. Then, a memory profiling is executed to obtain the

memory occupation. Finally, the application is modified to program the register file with the results from the profiling.

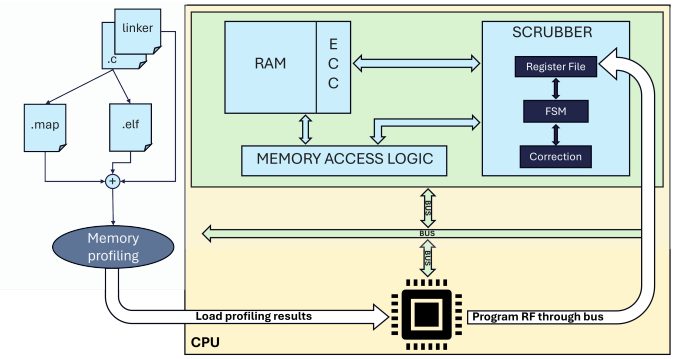

Fig. 1: Proposed Methodology

In the following subsections, the hardware architecture and the memory profiling are described in detail.

A. Hardware Design

A block diagram of the hardware design is shown in Figure 2.

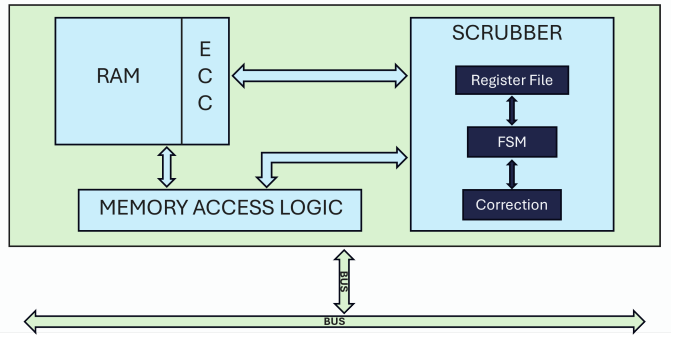

Fig. 2: Scrubber IP Block Design

Describing the design from the ground up, the IP is constructed around the RAM in use, making it agnostic to the specific type of memory, provided it supports ECC. This memory is then linked to a logic module that establishes the arbitration mechanisms between the scrubber and the core to access the memory, rendering the design compatible with single-port RAM configurations. This feature results in a significant reduction of logic element usage by 68%.
The scrubber instead follows a behaviour defined by an internal FSM. The module needs to know, prior to synthesis, the size of the RAM. The latter is then divided into slices of equal size, which is also defined at the synthesis stage. The RAM slices are mapped to a bit in the register file, whose size is:

$$\left\lceil \frac{Memory\_Length}{Memory\_Slice} \right\rceil$$

The scrubber executes as explained in Algorithm 1

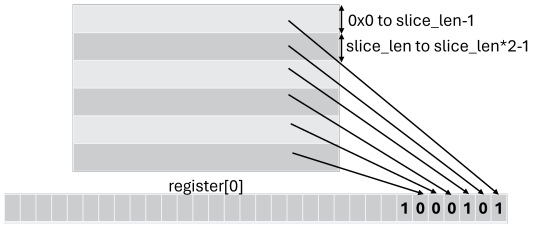

Fig. 3: Memory slicing

Algorithm 1 Scrubber algorithm

```
if enable_scubber then
    for counter ← 0 to Memory_Length do
        register_index ← ⌊address ÷ 32⌋ mod 32
        if Register[register_index] == 1 then
            read_memory(counter)
            counter ← counter + 4
        else
            counter ← counter + Memory_Slice
        end if
    end for
end if
```

The enabling signal can be derived from the addressable space as an additional register or deactivated by an internal watchdog that monitors memory accesses from the bus. This approach avoids race conditions by prioritizing memory load/store operations when they occur, ensuring seamless functionality. Two additional registers are used as counters, incrementing when single and double errors in the ECC are found. They can be reset by performing a write.

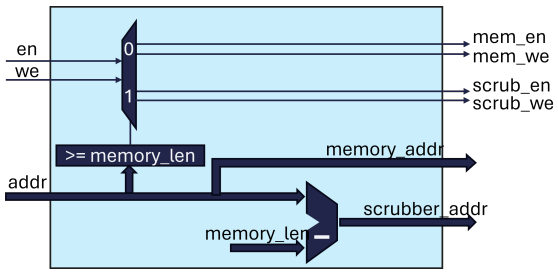

Fig. 4: Memory access logic

The access from the bus is controlled, as aforementioned, by a memory access logic module (Fig. 4). It is a purely combinational module that receives the control signals from the bus and contains multiplexers to forward them to the right submodule. The registers contained in the scrubber can be accessed by performing operations on the addresses succeeding the length of the RAM (i.e., if the memory is 1024 bytes in length, if a read operation is performed on address 1024, the scrubber's register at address zero is being accessed).
A top-level module encapsulates the scrubber, RAM and memory access logic into a single module which can be instantiated in a design, either by using an AXI Bus, or by

completely replacing an already present memory. This design can be used for whichever type of random access memory (e.g. Data RAMs or Instruction RAMs).

B. Memory Profiling

This section describes how firmware engineers can identify the active regions of RAM used by the application to configure the IP. The values of the registers in the IP can be determined directly by the firmware engineer, if he chooses to do so. Nevertheless, this method does not take full advantage of the improvements made by the architecture described in Section III-A. For this reason, an approach based on memory profiling has been developed to pre-process the application that will run on the core. The initial assumption made is that the code must be designed following the approach of predictability, having to be run on a safety-critical system [13], [14].

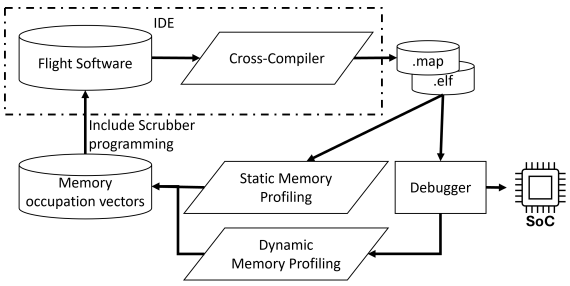

Fig. 5: Memory profiling

This methodology takes as input the input and output files from compilation. It then performs a search for all the variables stored in the RAM regions and their length using the elf file. The second stage is to execute the application on the device using a debugger. The debugger then looks for all the dynamic memory allocations performed, and stores in a log file the location and the size of the variable. As per the assumption made before, all the dynamic memory allocation must be made before the beginning of the main program, only during startup. Finally, the two log files generated are post-processed and the register content is given as a result. At this stage, the application can be modified to include register programming and scrubber enabling. The profiling procedure is described in Algorithm 2.

IV. Experimental Results

In the following subsections, a description of the experimental setup is presented, with the proposed solution tested both in simulation using a RISC-V microcontroller and synthesized on an FPGA.

A. Experimental Setup

The proposed experimental setup features the Pulpino RISC-V microcontroller [15]. It is an open-source single-core microcontroller system, based on a 32-bit RISC-V core "RISCY" [16] developed at ETH Zurich.

The design has been properly modified so that it does not use anymore the DRAM from the original design, but instead

Algorithm 2 Memory profiling algorithm

$elf\_file \leftarrow readelf\ \ flight\_software.elf$
$(variable,\ address) \leftarrow variables\_from\_elf(elf\_file)$
for $i = 0;\ i < sizeof(variables);\ i++$ **do**
  if $address \geq ram\_init\ \ \&\ \ address < ram\_end$ **then**
    $static\_profiling_addr.append(address)$
    $static\_profiling_size.write(size)$
  end if
end for
$start\_debugger(flight\_software.elf)$
$break\ mem\_alloc\_func$
while $program\_counter! = synch\_address$ **do**
  $continue\_execution()$
  if $return\_reg \geq ram\_init\ \ \&\ \ return\_reg < ram\_end$ **then**
    $dynamic\_profiling\_addr.append(return\_reg)$
    $dynamic\_profiling\_size.write(size\_reg)$
  end if
end while
$compute\_occupation\_vectors(vectors\_file)$

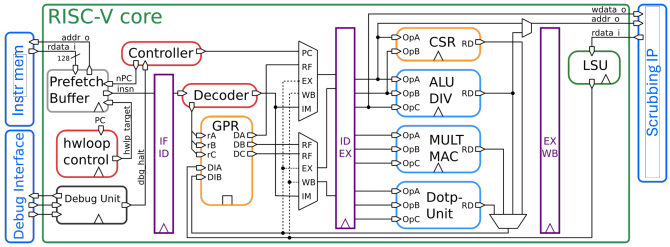

Fig. 6: Experimental Setup featuring Pulpino

it has been remapped to use the proposed design, as shown in Figure 6, featuring the scrubber with a custom RAM with 7 bits of ECC. The core has also been modified to go from 32kB to 1MB of DRAM and 64kB of IRAM. The linker was modified as well to include bigger data and stack sections, and a third section called "scrub" which includes the memory addresses of the scrubber's register file.

The design has been synthesized on a ZCU104 from AMD [17]. Instead of the custom RAM, a BRAM [8] has been used with 8 bits of ECC. In Table I a brief overview of the utilization on the FPGA is shown.

For both setups, the tests have been performed using a simple ECC-based RAM, a classic scrubber, and finally the proposed architecture.

B. Test Applications

To test the design, multiple applications from the pulpino suite of programs have been used. Mostly the more expensive ones in terms of RAM usage, such as the ones based on matrix multiplication and machine learning tasks. For the purposes of this work, we present results based on matrix multiplication and the memory-hungry CNN algorithm. This

TABLE I: FPGA Utilization for a 2kB RAM

Type	RAM Type	Scrub Time	LUT	FF	Power Consumption
RAM with ECC	Single Port	N/A	5265	3204	1 mW
RAM with basic scrubber [10]	Double Port	$\frac{RAM\_Size}{f\_clk}$	8860	4870	3 mW
RAM with advanced scrubber (32w slices)	Single Port	$\frac{RAM\_Occupation}{f\_clk}$	6544	3597	3 mW

suite of programs, however, gives just an overview of the possible operations the avionic system could perform, but does not represent any actual mission scenario.

For this reason, we have decided to use an application already proven in relevant mission profiles, such as the ones from the ARDITO Rover [18], [19]. ARDITO is a technological demonstrator of a Rover for human assistance developed at Politecnico di Torino by the Student Team DIANA. The code has been ported from RTEMS to FreeRTOS and built to run on the Pulpino microcontroller. The ported code has the function to allow the mobility operations of ARDITO, receiving commands via CAN Bus from the Main On-Board Computer (MOBC), computing the heading of the 4 steering joints, and speed for each of the 6 wheels of the Rover through a dedicated, Ackermann-based controller, and sending them via a second CAN Bus. An overview of the tasks is in Figure 7 It also collects data from multiple sensors. The exchange of data through communication peripherals has been simulated in this scope.

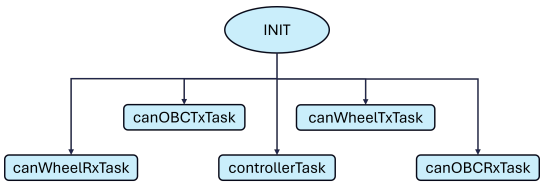

Fig. 7: ARDITO Navigation Firmware Tasks

The relevance of the test is given by the fact that it's an actual mission scenario for a planetary rover, giving the system long times of idle while the Rover may be performing operations with other payloads; interrupted by navigation scenarios, where the memory usage is more intensive.

C. Fault Injection Campaign

The fault injection campaign has been performed by automating the whole scenario, from elaboration of the design to fault generation and simulation. Once the application is compiled and profiled, a script to generate the SEUs and the time of injection is called. This script was designed to simulate and analyze Single Event Upset (SEU) occurrences in a deep space environment for 16nm technology, simulating the usage of a ZCU104. The primary objective was to establish a worst-case scenario baseline for SEU rates in deep space.
The script integrates a mathematical model for SEU generation and fault injection, leveraging validated scaling factors and experimental parameters. The simulation operates as follows:

- The memory array is modeled as a 1MB memory space, with each word consisting of 32 bits. Profiling data is used to identify valid memory locations for SEU injection, ensuring that faults are only injected into used words.
- The SEU rate is computed starting from the data recovered from the Cassini mission [20], which had a rate of 280 flips/day. Due to the lower technology size (16nm against ≈350nm) the SEU rate will worsen due to a number of factors. Using the CREME96 modeling [21], we can understand that the amount of critical charge (Q_crit) to flip a bit will decrease. Thanks to scaling, the sensitive volume of the cell is also smaller. While this reduces the probability of a high-energy particle interacting with the cell, it also makes the cell more vulnerable to low-energy particles, which are more abundant in space. Finally, due to the lower operating voltages, the radiation sensitivity increases. [22]
- SEU events are distributed temporally using a Poisson process, with probabilities calculated for various time windows. Spatial distribution is modeled based on deposited charge values and their corresponding SEU probabilities, reflecting the sensitivity of the memory cells to radiation-induced charge deposition.
- The simulator generates SEU events by randomly selecting valid memory locations and injecting faults at specific bit positions. Each event is timestamped and formatted into commands for injection into the memory. The number of events is determined by the Poisson-distributed SEU rate over the simulation duration.

A simulation campaign has been launched for each setup. Each test application generates a fault report, which contains both the number of detected SEUs and the mean time between the SEU happening and it being detected by the logic (in clock cycles).

Table II shows the results collected from the aforementioned experiments. The three applications being run are the ones described in Section IV-B. The memory occupation has been computed as the sum of all the occupied words in memory, using the profiling results, plus the space reserved for the stack memory. The first test performs a multiplication between matrices. This test uses a lot the memory and always in the same sections. All three designs are able to capture the errors, but the mean time between the injection and the correction of

TABLE II: Experimental Results

Application	Memory Occupation	SEU Captured/Injected			Mean Time injection to correction (cc)		
		ECC	Basic Scrub	Adv Scrub	ECC	Basic Scrub	Adv Scrub
Matrix Multiplication	6.40%	31/31	31/31	31/31	88354	73317	24124
CNN	34.96%	70/73	73/73	73/73	352852	175634	45184
ARDITO Code	25.57%	16/94	94/94	94/94	3544	156728	26018

the errors is far lower in the proposed scrubber. In the second case, the memory occupation reaches almost 35%. Both the simple scrubber and the proposed one are able to catch all the injected errors, but the main difference can be seen in the clock cycles required to correct a fault, which is almost 4 times as much for the basic scrubber.

The third test, however, is more adherent to a real mission scenario. The low results of the ECC-only test are due to the fact that the program changes its behaviour based on the input values, which, during the simulation time, didn't reach the section of the algorithm that works on those specific variables. As in the other tests, the performances of the advanced scrubber are far higher than the basic.

V. Conclusions

The increasing interest in the space sector, as well as the shift from government-sponsored missions to the private sector, has brought the need to bring mission costs down to accelerate access to space and to the surface. Because of this, miniaturization of platforms has become a need for companies, moving also the avionics technology from space-grade to high-performance COTS components.

The direct consequence of this paradigm shift is the increasing necessity to mitigate the radiation-induced effects via a mix of Hardware and Software mitigation techniques.

The presented work aims at introducing a more efficient scrubbing mechanism using an application-profiled RAM technique. The proposed scrubber is capable of reducing the scrubbing time by checking only the used portion of the memory, without wasting time on empty cells. It also has an arbitration mechanism which allows it to be used with a single-port RAM, decreasing the occupation by more than $\approx 50\%$ with respect to the classical solutions. Moreover, the register file of the IP can be programmed and activated at any moment in time, allowing on-line reprogramming for even more efficient scrubbing. Experimental results show how the scrub time is reduced by a factor which increases as memory usage decreases.

A. Future Works

Future work will focus on developing a more complex profiling script to analyze variable usage, enabling the creation of a detailed 2D map of memory read and write operations. This enhanced mapping will facilitate targeted scrubbing by monitoring only the stack memory up to the stack pointer and using a more performant scrubbing based on the amount of operations done on each variable in memory, optimizing resource utilization.

References

[1] P. Michel and M. Kueppers, "The science return of the esa hera mission to the binary asteroid didymos," in *European Planetary Science Congress*, 2020, pp. EPSC2020–88.

[2] J. Bellerose *et al.*, "Double asteroid redirection test (dart): navigating to obliteration," *Acta Astronautica*, vol. 219, pp. 417–427, 2024.

[3] K. A. Farley *et al.*, "Mars 2020 mission overview," *Space science reviews*, vol. 216, no. 8, 2020.

[4] K. D. Prasad *et al.*, "Chandrayaan-3 alternate landing site: Pre-landing characterisation," *arXiv.org, 2023*, 2023.

[5] B. Yost and S. Weston, "State-of-the-art small spacecraft technology," NASA, Tech. Rep., 2024.

[6] E. Normand, "Single-event effects in avionics," *IEEE Transactions on Nuclear Science*, vol. 43, no. 2, pp. 461–474, 1996.

[7] E. Stassinopoulos and J. Raymond, "The space radiation environment for electronics," *Proceedings of the IEEE*, vol. 76, no. 11, pp. 1423–1442, 1988.

[8] AMD, https://docs.amd.com/v/u/en-US/pg058-blk-mem-gen.

[9] A. Pérez-Celis, C. Thurlow, and M. Wirthlin, "Identifying radiation-induced micro-sefis in sram fpgas," *IEEE Transactions on Nuclear Science*, vol. 68, no. 10, pp. 2480–2487, 2021.

[10] R. Glein *et al.*, "Bram implementation of a single-event upset sensor for adaptive single-event effect mitigation in reconfigurable fpgas," in *2017 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, 2017, pp. 1–8.

[11] ——, "A self-adaptive seu mitigation system for fpgas with an internal block ram radiation particle sensor," in *2014 IEEE 22nd Annual International Symposium on Field-Programmable Custom Computing Machines*, 2014, pp. 251–258.

[12] J. Chen *et al.*, "Design of sram-based low-cost seu monitor for self-adaptive multiprocessing systems," in *2019 22nd Euromicro Conference on Digital System Design (DSD)*, 2019, pp. 514–521.

[13] R. DO, "Rtca do-178c, software considerations in airborne systems and equipment certification," *Inc., December*, 2011.

[14] MIRA Ltd, *MISRA-C:2004 Guidelines for the use of the C language in Critical Systems*, MIRA Std., Oct. 2004. [Online]. Available: www.misra.org.uk

[15] A. Traber *et al.*, "Pulpino: A small single-core risc-v soc," in *3rd RISCV Workshop*, 2016, p. 15.

[16] M. Gautschi *et al.*, "Near-threshold risc-v core with dsp extensions for scalable iot endpoint devices," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 10, pp. 2700–2713, 2017.

[17] Xilinx, "Zynq ultrascale+ mpsoc zcu104 evaluation kit," https://www.xilinx.com/products/boards-and-kits/zcu104.html/.

[18] L. Caraccio *et al.*, "Ardito, a modular technology demonstrator for robotic planetary surface exploration and operational support: an overview," *IEEE Aerospace and Electronic Systems Magazine*, 2025.

[19] N. di Gruttola Giardino *et al.*, "A modular avionics architecture for a planetary rover demonstrator for human assistance," vol. 58, no. 16, 2024, pp. 181–186, 2nd IFAC Workshop on Aerospace Control Education - WACE 2024.

[20] H. Garrett *et al.*, "Analysis of single-event upset rates on the clementine and cassini solid-state recorders," *Journal of Spacecraft and Rockets - J SPACECRAFT ROCKET*, vol. 47, pp. 169–176, 01 2010.

[21] A. Tylka *et al.*, "Creme96: A revision of the cosmic ray effects on micro-electronics code," *IEEE Transactions on Nuclear Science*, vol. 44, no. 6, pp. 2150–2160, 1997.

[22] A. Constante, T. Balen, V. Champac, L. Poehls, and F. Vargas, "Impact of aging on the seu immunity of finfet-based embedded memory systems," *Microelectronics Reliability*, vol. 150, p. 115229, 2023, special issue of 34th European Symposium on Reliability of Electron Devices, Failure Physics and Analysis, ESREF 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0026271423003293