Dependable Computing for Spacecraft

Christian M. Fuchs*†, Nadia M. Murillo†, Aske Plaat*, Erik van der Kouwe*, Daniel Harsono†, and Peng Wang*
*Leiden Institute for Advanced Computer Science †Leiden Observatory;
Leiden University, The Netherlands email: christian.fuchs@dependable.space

***Abstract*—In this paper, we provide insights on the practical feasibility, effectiveness, and validation of the multi-stage fault-tolerance architecture. We exploit thread-level coarse-grain lockstep to facilitate forward-error-correction and assures computational correctness on an FPGA-based MPSoC. It is ITAR-free and can be implemented using commercial hardware with standard OS such as FreeRTOS, RTEMS, or Linux.**

I. Introduction

Modern embedded and mobile-market hardware is a driving factor in satellite miniaturization, enabling a smaller, lighter, and cheaper class of spacecraft. Small satellites (<100kg) today can offer similar performance as large spacecraft, but are less complex and can be developed rapidly with low manpower and even student teams. They are, thus, popular for a variety of commercial and scientific undertakings, enabling space missions which were technically infeasible just a decade ago. However, these spacecraft suffer from low reliability, discouraging their use in long or critical missions.

The space environment poses many challenges: radiation can physically damage semiconductors and induces a variety of faults, from single bit-flips, to component-wide functional interrupts, and spontaneous aging [1]. Faults must be handled fully autonomously, as physical access even in LEO is impossible and the limited link-bandwidth and high latency constrains failure analysis severely.

Therefore, various fault tolerance (FT) concepts have been developed for large spacecraft which facilitate radiation hardness through hardware redundancy, voting, and specialized semiconductor manufacturing in large feature-size. This approach however is ineffective for modern systems-on-chip (SoCs) manufactured, and there is also no known singular FT techniques which can offer sufficient fault-coverage and long-term robustness in deep space or earth orbit.

Instead, a combination of different FT techniques is necessary which can upkeep fault-coverage and assure computational correctness, memory integrity, availability and safety. For miniaturized spacecraft an on-board computer must therefore assure guarantee dependable operation when critical faults occur, instead of attempting to assure fault-immunity through additional hardware FT. Hence, we are developed a dependable on-board computing architecture which utilizes software-FT, erasure coding, topological features and error scrubbing to guarantee these properties even in a degraded system.

II. Our Dependable Computer Architecture

We developed our architecture for use within an FPGA, and utilize exclusively commercial-off-the-shelf (COTS) hardware to deliver low-cost fault-tolerance for nanosatellites. We realized this architecture as MPSoC design, depicted in Figure 2. The high-level logic of this approach is depicted in Figure 1, and consists of three interlinked fault mitigation stages implemented across the embedded stack:

Stage 1 implements forward error correction and utilizes thread-level coarse-grain lockstep to generate a distributed majority decision across a set of isolated, weakly couple processor cores. Fault detection is facilitated through application-provided callback functions, requiring no modifications to an application or knowledge about intrinsics. Faults are resolved through state re-synchronization and thread migration to tiles with spare processing capacity. Stage 1 is described in detail in [2], in which we also established an upper bound for the performance cost of the lockstep.

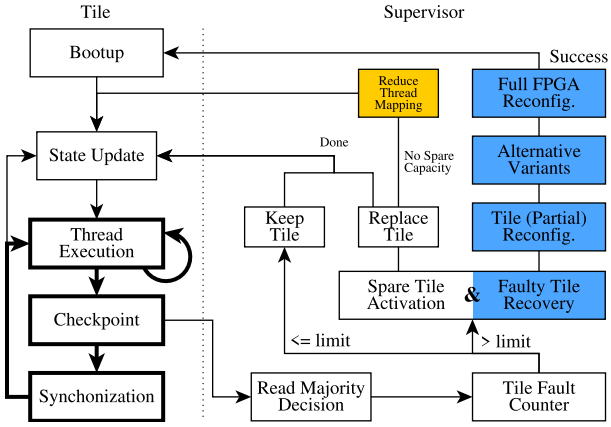

Fig. 1: Stage 1 (white) assures fault detection (bold) and fault coverage, Stage 2 (blue) and 3 (yellow) counter resource exhaustion.

Stage 2 recovers tiles through FPGA reconfiguration, thereby counteracting resource exhaustion. It assures the integrity of the FPGA's running configuration and deploys scrubbing as well as Xilinx SEM to correct transients in FPGA fabric. Its objective is to repair defective tiles affected by upsets in tile logic, and to cover permanent faults using alternative configuration variants. Individual stand-alone concepts utilizing comparable mechanics as Stage 2 have been researched and validated e.g. by Nguyen et al. in [3]. FDIR concepts using configuration scrubbing and partial reconfiguration have also been demonstrated on-orbit [4].

Stage 3 is activated when too few healthy tiles are available due to accumulation of permanent faults, and **re-allocates processing time** to maintain reliability. To do so, it utilizes the thread-level **mixed criticality** found in satellite computing, assuring sufficient compute resources are available to high-criticality applications by sacrificing performance of lower-criticality threads. This functionality as well as our architecture's adaptive capabilities are presented in [5].

The MPSoC consists of independent processor tiles equipped with a processor's immediate peripherals (e.g., interrupt controller), external interfaces, and a supervisor access port. As on-chip memory alone is insufficient for all but trivial applications, tiles utilize a shared set of memory controllers, which are implemented redundantly to enable load-balancing and safeguard from radiation-induced functional interrupts. Other interfaces such as CAN, SPI, I2C are thus replicated for

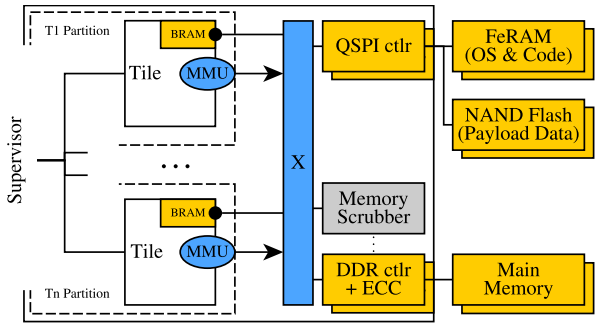

Fig. 2: The topology of our tiled MPSoC design. Each tile exists in its own reconfiguration partition, simplifying routing and clocking.

Impact	Fault Detectable	Detectable by victim tile	Detectable by other tiles	Recovery through	Observed Effect per Fault Type: Transient	Permanent	Intermittent
Corrupted State	yes	yes	yes	state-update	49%	44%	53%
Thread Crash	yes	yes	no	state-update	8%	17%	10%
Lockstep Failure	yes	no	yes	reboot	1%	2%	1%
OS Crash	yes	no	yes	reboot	10%	18%	15%
Masked (no effect)	(some*)	(yes*)	(no*)	(reboot*)	32%	19%	21%

Tab. I: Fault injection experiment results for our RTOS implementation divided into transient, permanent, and intermittent faults. Masked faults affecting OS data structures could be detected through OS-level EDAC, which not are in place in our current RTEMS proof-of-concept.

each tile, making them indistinguishable from a software perspective. It is not viable to instantiate one external DDR memory controller per tile due to their large footprint, package-pin and PCB space limitations. This allows live-migration of application threads between tiles, enables code reuse, and minimizes the footprint.

III. Implementation and Validation Results

The precise effects of radiation on semiconductors induced by a fault vary. They depend on the particular effected chip-region, logic, and microfabrication technology used [1], and today and the only practical way to evaluate them is radiation testing or approximation using laser fault injection. However, this can occur only at a very late stage in development, and would be unsuitable to verifying the effectiveness of a software-based FT mechanic as the radiation testing is itself not systematic. Dependable software systems therefore are best evaluated using fault-injection, as this allows sufficient test coverage while maintaining realism.

We implemented Stage 1 and 3 in RTEMS 4.11.2, which is a real-time OS used in a broad variety of space applications, from satellite control to instrumentation and ground-support equipment. As payload application protected by our implementation, we utilized ESA's Next Generation DSP benchmark and the application we had utilized for overhead estimation in [2].

Our architecture can be implemented using COTS hardware and extensively validated, and widely available library IP, requiring no proprietary logic or space-grade processor cores. The processor architecture considered in our project is the mobile-market ARM Cortex-A53, though a functionally equivalent design was also developed using Xilinx Microblaze/RISC cores. This is possible was implemented on a variety of Xilinx Ultrascale FPGAs (4 core utilization on XCKU5P of 28% LUTs, 33% BRAMs, 16% FFs, 5% DSPs with 1.92W total power consumption), see also [2].

In practice, choosing the right test-space for a practical OS-scale implementation is non trivial, in contrast to what is described as ideal in literature. Sufficient test coverage for such software can often be unobtainable in practice, and even fault injection using state-of-the-art tools requires a compromise between realism and test-coverage to avoid runaway test-times and extreme equipment. Besides test coverage, our architecture has to cope with not merely transient faults, but also radiation-induced permanent faults, as these are common in modern memories and processor components flying in space.

We chose to utilize ISA-level fault injection to inject transient and permanent faults, and simulate regional functional interrupts, as several powerful fault-injection tools have emerged in recent years. While most of these tools are proprietary, or closed-source and unavailable to the public [6], [7], the two frameworks FAIL [8] and FIES [9] are available publicly as open source. We utilize FIES, as it was developed specifically to validate ARM-based COTS critical systems, while FAIL mainly targets the ia32/amd64 platforms. In the process of developing our test toolchain, we also extended FIES to support different tracing techniques and added functional improvements[1]. Table I contains a summary of results of our fault-injection experiment.

IV. Conclusions

In this paper, we provide insights on the practical feasibility and effectiveness of the multi-stage fault-tolerance architecture we first presented in [2]. We constructed a proof-of-concept implementation and evaluated its fault detection and recovery capabilities using ISA-level fault injection into registers, memory and a tile's processors pipeline. To our knowledge, this dependable computing architecture is the first practical, non-proprietary, affordable solution offering FT general-purpose computing aboard nanosatellites.

Our architecture utilizes fault tolerance measures across the embedded stack, and combines topological with software-side functionality, utilizing only of extensively validated standard parts. Thereby, we enable enable the use of nanosatellites in critical space missions, while the architecture allows the trading processing capacity for reduced energy consumption or fault-coverage. The architecture was implemented successfully, and tested on current generation Xilinx FPGAs with 4, 6 and 8 tiles, and validated through fault-injection into a practical real-time OS based implementation using state-of-the-art open-source tools.

Based on our proof-of-concept, we assert that our architecture is effective and does not exceed the tight energy, complexity and cost constraints of even very small spacecraft such as CubeSats. The positive outcome of the fault-injection campaign enables us to develop a hardware prototype OBC to test resilience of our architecture at the system-level using laser fault injection and radiation testing.

V. Acknowledgements

This research was developed for an ESA/NPI project, and we thank our peers at TEC-EDD for their support. We thank ARM Ltd. for making available the relevant IP. N.M.M. and D.H. acknowledge funding through A-ERC grant 291141, NOVA, and KNAW.

References

[1] J. Schwank *et al.*, "Radiation Hardness Assurance Testing of Microelectronic Devices and Integrated Circuits," *IEEE Transactions on Nuclear Science*, 2013.
[2] C. M. Fuchs *et al.*, "Bringing fault-tolerant gigahertz-computing to space," in *IEEE ATS*, 2017.
[3] N. T. H. Nguyen, "Repairing FPGA configuration memory errors using dynamic partial reconfiguration," Ph.D. dissertation, The University of New South Wales, 2017.
[4] D. Petrick *et al.*, "Adapting the reconfigurable spacecube processing system for multiple mission applications," in *IEEE Aerospace Conference*. IEEE, 2014.
[5] C. M. Fuchs *et al.*, "Dynamic fault tolerance through resource pooling," in *NASA/ESA AHS*. IEEE, 2018.
[6] R. Natella, D. Cotroneo, and H. S. Madeira, "Assessing dependability with software fault injection: A survey," *ACM Computing Surveys*, 2016.
[7] A. Johansson, "Robustness evaluation of operating systems," Ph.D. dissertation, TU Darmstadt, 2008.
[8] H. Schirmeier *et al.*, "FAIL: An open and versatile fault-injection framework for the assessment of software-implemented hardware FT," in *EDCC*. IEEE, 2015.
[9] A. Höller *et al.*, "FIES: a fault injection framework for the evaluation of self-tests for COTS-based safety-critical systems," in *MTV*. IEEE, 2014.

[1] Our patches are available at https://fieser.dependable.space