Towards Affordable Fault-Tolerant Nanosatellite Computing with Commodity Hardware

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

***Abstract*—Modern embedded and mobile-market processor technology is a cornerstone of miniaturized satellite design. This type of lighter, cheaper, and rapidly developed spacecraft has enabled a variety of new commercial and scientific missions. However micro- and nanosatellites (≤100kg) currently are not considered suitable for critical, high-priority, and complex multi-phased missions, due to their low reliability. The hardware fault tolerance (FT) concepts used aboard larger spacecraft can usually not be used, due to tight energy and mass constraints, as well as disproportional costs. Thus, we developed a hardware-software hybrid FT-approach, which enables FT through software-side coarse-grain lockstep, FPGA reconfiguration, and thread-level mixed criticality. This allows our FPGA-based proof-of-concept implementation to deliver strong fault coverage even for missions with a long duration, but also to adapt to varying performance requirements during the mission. In this paper, we present the implementation results on a tiled multiprocessor system-on-a-chip (MPSoC) design we developed as an ideal platform for our approach. We provide details on the validation of our approach through fault injection, which show that our lockstep implementation is effective and efficient for providing FDIR within our system, and show in direct comparison that our results are consistent with related work. These results show that our architecture is effective, overhead efficient, and remains within the tight energy, complexity, and cost limitations of even very small spacecraft such as CubeSats. To our knowledge, this is the first fault mitigation approach offering strong fault tolerance, which can uphold computational correctness viable for miniaturized spacecraft and is not dependent on proprietary processor cores.**

I. Introduction

Modern embedded technology is a driving factor in satellite miniaturization, enabling a smaller, lighter, and cheaper class of spacecraft, fueling a massive boom in satellite launches and a rapidly evolving new space industry. These micro- and nanosatellites (≤100kg wet mass) have become increasingly popular for a variety of commercial and scientific missions. However, they suffer from low reliability, discouraging their use in long or critical missions, and for high-priority science.

For larger spacecraft, various hardware-based fault tolerance (FT) concepts are available, which are not common aboard miniaturized spacecraft due to tight energy, mass, and volume constraints, and disproportional costs. Conventional embedded and mobile-market systems-on-chip (SoCs) are deployed in their stead, which usually lack even basic FT functionality. In consequence, a significant share of post-deployment failure of nanosatellites can be attributed directly to the failure these components and peripheral electronics [1].

Therefore, we developed a non-intrusive, flexible, hybrid hardware/software approach to assure FT with COTS mobile-market technology, which is described in detail in [2]. The MPSoC consists of well tested COTS components, library logic (IP), and powerful mobile-market processor cores, yielding a non-proprietary architecture. In this paper, we present practical implementation results of this approach made with our MPSoC design, provide first fault injection experiment results, and compare these to the coarse-grain lockstep experiments conducted by Dobel at al. in [3]. These result show that our architecture is effective, and does not exceed the tight energy, complexity and cost constraints of even very small spacecraft such as CubeSats with scientific or commercial payload.

The following two sections contain background information in our field of application, information on the fault types our approach must handle, and a discussion of related work. In Section IV a brief overview over the three stages of our approach is provided. We describe our proof-of-concept MPSoC-design in Sections V and VI, and practical implementation details are provided in Section VII. Section VIII contains early validation results of our coarse-grain lock-step approach, which were obtained using fault injection, followed by future work.

II. The Space Environment & Radiation

The drastically different fault-model [4] and form factor constraints [5] prevent the re-use of many FT and testing approaches developed for ground applications. Even in atmospheric aerospace applications, these usually consider availability, non-stop operation, and safety, but rarely computational correctness for a fully autonomous system.

Physical access to a satellite after deployment today is in practice impossible, and historical servicing missions were conducted only on rare occasions for satellites of outstanding importance in low-Earth orbit (LEO). Signal travel times, limited communication windows, and scarce bandwidth make live interaction with a spacecraft impractical. Thus, faults detected by our approach are resolved autonomously during a satellite mission (10+ years).

High-energy particles are the predominant cause of faults [6] during a satellite mission. They travel along the Earth's magnetic field-lines in the Van Allen belts, are ejected by the Sun during Solar Particle Events, or arrive as Cosmic Rays from beyond our solar system. These particles can corrupt logical operations or induce bit-flips within memory and semiconductor logic (single event effects - SEE), and may cause displacement damage (DD) at the molecular level to a chip's substrate and circuitry. The energy threshold above which SEEs induce transient faults in chips manufactured in fine technology node decreases, and the ratio of events inducing multi-bit upsets (MBU) or permanent faults increases. Radiation events can also cause single event functional interrupts (SEFIs), affecting sets of circuits, individual interfaces, or even entire chips.

In general, the effects of bit-upsets and SEFIs can be transient or permanent, while DD is always permanent [4]. The accumulative nature of permanent faults implies accelerated and often spontaneous aging, which must be handled efficiently throughout a mission. The cumulative effect of charge trapping in the oxide of electronic devices (total ionizing dose – TID) further impacts the lifetime of an on-board computer (OBC). However, chips manufactured in certain new technology nodes, such as recent generation FPGAs [7] show drastically better than expected TID performance [8] and resistance to latch-up [9] in contrast to projections based on technology scaling.

In LEO, the residual atmosphere and Earth's magnetic field provide some protection from radiation, but this absorption effect diminishes quickly with distance. Many miniaturized spacecraft operate in this region, and forego FT in favor of developing a functional satellite within the boundaries of their limited resources and manpower. Most nanosatellites today thus utilize COTS microcontroller- and application processors-SoCs, FPGAs and combinations thereof [10], [11], occasionally introducing basic, custom-designed redundancy with ground-triggered fail-over. However, this is not an option for critical and long-term missions with a scientific or commercial objective, as these components may fail at any given point in time, resulting in a loss of mission. Most nanosatellite hardware development efforts are more comparable to rapid hardware-prototyping than to the sophisticated and thorough ASIC development process. FPGAs have, hence, become increasingly popular as they are well suited for this design approach, despite being more vulnerable to radiation than ASICs, due to their better FDIR potential [12].

III. Related Work

FT is traditionally implemented through circuit-, RTL-, core-, and OBC-level majority voting [13]–[15] using space-proprietary IP, which is difficult and costly to maintain and test. Circuit-, RTL-, and core-level voting are effective for small SoCs such as microcontrollers [16]. More complex application processor designs would require vast arrays of synchronized high-frequency voters to facilitate core-lockstep in hardware. Hence, core-level lockstepping at GigaHertz clock rates is non-trivial and has been implemented only at low frequencies, even with chips manufactured in modern technology nodes [17]–[20]. Software takes no active part in fault-mitigation in these concepts, as faults are suppressed at the circuit level, preventing the effective assessment of a processor's health beyond hardware fault counters. In general, hardware-voting based MPSoC designs are non-adaptive, as a design's fault coverage properties are design time specific and usually also chip dependent [21].

FPGA-based systems can offer excellent FDIR potential [12], as transients in major parts of the FPGA fabric can be corrected and even permanent faults may be compensated through reconfiguration with differently routed configuration variants [22]. Fine-grained, non-invasive fault detection in FPGA fabric, however, is challenging, and subject of active research [23], [24]. Hence, FT-concepts for programmable logic resort to error scrubbing, which has scalability limitations and covers only parts of the fabric [23], [25]. We therefore facilitate fault detection in software using coarse-grain lockstep.

Software-based FT measures have been shown to be effective for modern embedded hardware [3], [26], [27], but have largely been ignored by the space industry. While many of them include innovative ideas, they exist only in theory and leave fundamental practical issues unsolved. Most implement checkpoint & rollback or restart, which makes them unsuitable for spacecraft command & control applications [28], others ignore fault detection [29], [30], or require infallible fault detection entities with deep knowledge about application-intrinsics at runtime [31] without considering how such knowledge could be obtained. Faults often are assumed to be isolated, side effect free and local to an individual application thread [32] or transient [3], [29], and entail high performance [33] or resource overhead [34], [35]. As shown in [2] these limitations can be elegantly overcome by combining forward error correction and thread-level lockstep within a generic MPSoC with three or more independent processor cores, and can offer long-term fault coverage in a tiled SoC topology.

Tiled architectures [36], [37] are often used for well paralellizable applications with many low-performance processor cores. Among others, [38] and [37] showed that such typologies exhibit a degree of inherent redundancy beneficial to FT, and but the relevant concepts were only compatible with image processing applications with a very specific structure. However, to implement sophisticated and efficient thread migration, fault detection must be facilitated at the operating system (OS) or application-level without falling back to design space exploration. Coarse-grain lockstep of weakly coupled cores can do just that, and we show fault injection results based on our approach [2].

IV. Our Software-Side FT Approach

The MPSoC described in this paper can offer strong FT using just COTS components and standard logic. This is made possible through the use of the FT approach we presented in [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 coarse-grain lockstep of weakly coupled cores to generate a distributed majority decision across tiles. Fault detection is facilitated through application callback functions, without requiring 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 2 recovers defective 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 recover from upsets in tile logic, and cover permanent faults using alternative configuration variants.

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

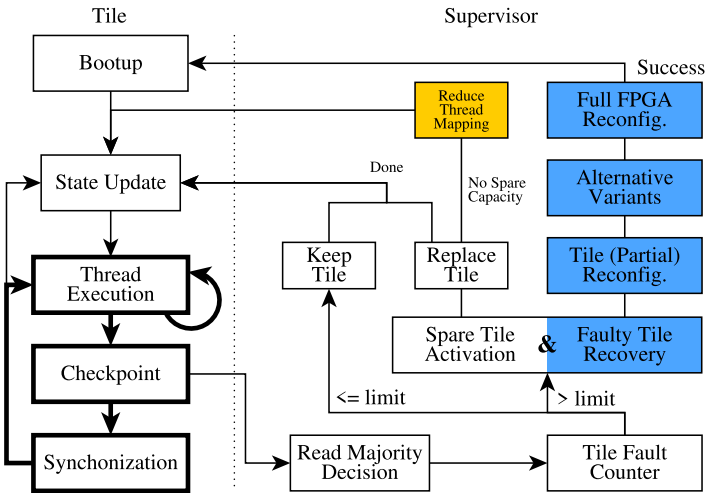

Fig. 1: Stage 1 (white) assures fault detection (bold) and fault coverage, Stage 2 (blue) and 3 (yellow) counter resource exhaustion and adapt to reduced system resources.

compute resources are available to high-criticality applications by sacrificing performance of lower-criticality threads.

Further details on this approach including benchmark results are available in [2], whereas its adaptive capabilities are presented in [39]. The main processor architecture considered in our project is the ARM Cortex-A53 application processor, as it is today widely used in embedded and mobile-market devices. However, this research is processor and ISA independent, and platform agnostic. In the next sections, we describe a publicly reproducible MPSoC sibling-variant of our design implemented using Xilinx Microblaze cores and Library IP implemented using Xilinx Vivado 2017.1 and later.

V. Tile Architecture

Our MPSoC architecture consists of multiple isolated SoC-compartments accessing shared main memory and OS code. Even though the purpose and function of these compartments is different, the topology resembles a tiled MPSoC rather than a conventional design, in which cores share infrastructure and peripherals. This topology maximizes Stage 1's fault coverage capacity and allows task mapping for opaque software.

Each such tile contains a processor core, local interconnect, and peripheral IP-cores and interfaces as depicted in Figure 2, resides in its own clock domain, and can be reset independently. Placing tiles in individual clock domains relaxes timing, minimizes interdependencies between tiles, enables partial reconfiguration and clock-gating.

A tile executes a set of thread replicas, and its loss can be compensated by the rest of the system. To assure a failed tile can not cause performance degradation in the rest of the system e.g. by continuously accessing DDR or program memory, it can be disconnected off the global interconnect by the supervisor. Non-masked faults (due to radiation, ageing, and wear) disrupt the data or control flow of software running on a tile (its state). Stage 1 builds upon this capability at the thread-level, as state difference can be detected, often by the malfunctioning tile [2].

Tiles are equipped with the same interfaces, with peripherals being mapped to identical address ranges. The tile address space layout is uniform across the system and tiles are indistinguishable for software. Hence, application code and data structures are portable between tiles, simplifying thread migration drastically. This allows us to reduce the computational cost and complexity of the lockstep.

Thread allocation and information relevant to the coarse-grain lockstep is stored in a dedicated dual-ported on-chip BRAM on each tile. One port is accessible to the tile's processor core,

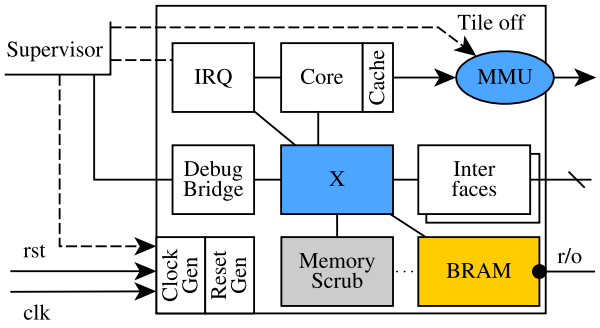

Fig. 2: Logic-side architecture of a tile with clocking and reset facilities. Access to local IP bypasses the cache, while access to global memory passes is cached for performance reasons.

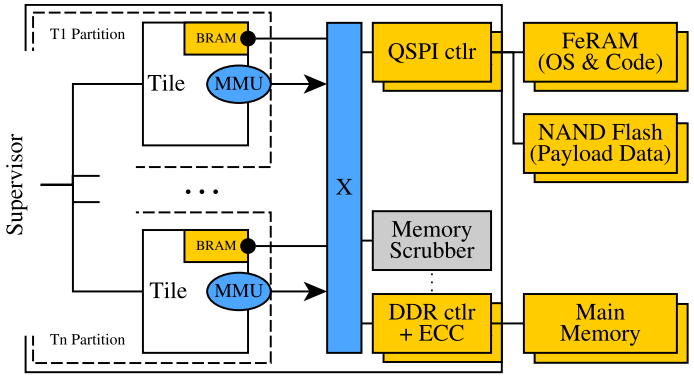

Fig. 3: The topology of our tiled MPSoC design. Each tile exists in its own reconfiguration partition and therefore also clock domain, simplifying routing.

while the other is read-only accessible to the system, allowing low-latency information exchange between tiles without cache-coherence or DDR access.

To allow supervisor access to a tile and its address space, each tile is equipped with a AXI debug-bridge. The supervisor can induce a reset and execute a self-test functionality run within a tile to detect faults in peripherals. It can also trigger an adjustment of the thread allocation in Stage 1 and 3, making the MPSoC's computational performance, robustness and energy consumption adjustable at runtime.

VI. Interconnect Topology & Shared Memory

Figure 3 depicts the MPSoC's high-level topology with clock domains, and supervisor access ports indicated. It includes an AXI interconnect in crossbar mode for access to main and non-volatile memory controllers, though we are reworking our MPSoC to use a NoC [38] instead.

Main memory is shared between tiles, as SD- and DDR memory controllers have a too large footprint to instantiate for each tile. Each tile has full access to a separate main memory segment, which contains the tile's stack and heap segments. It is mapped to the same address range on all tiles (the MMU component in the figures). All tiles can access main memory read-only to allow simple state synchronization and IPC between threads on different tiles.

To safeguard from SEFIs and permanent device failure, these memories and their controllers are implemented redundantly to enable fail-over. This also provides the hardware-side features necessary to enable application protective measures for non-volatile memory [40]. It also enables load distribution for main memory through segment interleaving.

VII. Practical Implementation Results

This MPSoC design was implemented on a Xilinx XCKU5P FPGA with modest resource utilization (28% LUTs, 33% BRAMs, 16% FFs, 5% DSPs) and 1.92W total power consumption. This implementation serves as proof-of-concept, and we are aware there is considerable potential for optimization. For comparison, the smallest popular nanosatellite form factor, 1U CubeSats[1], today offers an average power budget between 2W and 8W [41], these are commonly educational, short term missions without a reliability-critical scientific or commercial payload. However, in our research we consider reliability critical

[1]A CubeSat unit (U) designates to a $10cm^3$ cube with 1.33kg

spacecraft with a long-term mission, for which 2U and 3U CubeSats have become increasingly popular. 2U CubeSats offer a power budget of 15W+ due to increased solar panel surface, and even in its current early proof-of-concept, this architecture stays within the energy, complexity and cost constraints of our targeted application. We refrain from conducting an unfair comparison to radiation-tolerant or -hardened FPGAs used in classical space applications, as the even only the static power consumption of devices exceeds our implementation's total power due to 20 years of refined semiconductor manufacturing [42]. A detailed utilization report is available in [43].

In our proof-of-concept MPSoC, we can utilize FeRAM [44] or MRAM [45] connected via SPI as program memory, as both memory technologies are inherently radiation immune. This simplifies the design and reduces the footprint of the individual tiles, while taking advantage of the fact that the OS and program code can be shared between tiles due to their identical address spaces. Alternatively, program memory could be implemented independently for each tile on-chip or via SPI. This would imply the use of dedicated memories for each tile, increasing FT but inflating complexity and energy consumption.

Main memory in our proof-of-concept is implemented as DDR4 RAM due to easier migration from development boards to a custom PCB, as all Xilinx Ultrascale+ development boards are equipped with DDR4 memory. For nanosatellite missions to LEO, often only SECDED ECC support is required and available within existing Xilinx library IP, while basic EDAC scrubbing can be facilitated in software. For critical, deep-space, and long-term missions, block coding should be used instead to compensate for the increased impact of SEEs and higher likelihood of MBUs in highly-density SDRAM. Reed-Solomon ECC as well as AXI error scrubbers are available commercially, or can be assembled from open-source IP.

Tiles are placed in separate configuration partitions to enable partial reconfiguration of individual tiles, without affecting the rest of the system. We deployed configuration error mitigation through Xilinx Soft-Error-Mitigation for Ultrascale FPGAs (SEM) in combination with supervisor-side configuration scrubbing to safeguard logic integrity. To facilitate FPGA reconfiguration and transient fault scrubbing in the running configuration, an off-chip supervisor is utilized. As our multi-stage FT approach puts only minimal load on the supervisor, this can again implemented using a traditional radiation hardened microcontroller. The FeRAM-based MSP430 family would be a solid radiation-tolerant but non-FT substitute for an ultra-low-cost implementation of our approach in academic CubeSat projects, as the CubeSat community used microcontrollers of this family with encouraging results.

A fault resolved in Stage 1 may cause incorrect data to be emitted through I/O interfaces, an inherent limitation to coarse-grain lockstep [3] but acceptable for less critical nanosatellites. Larger spacecraft already utilize interface replications or even voting to protect I/O consistency, usually requiring considerable effort in hardware or logic to facilitate replication. Our MPSoC architecture inherently provides interface replication by design, and requires no extra measures.

Further safeguards are necessary for low-end CubeSats where interface replication is not desired, i.e., due to PCB-space constraints or to limit hardware complexity. Most embedded interfaces like I2C and SPI allow a simple majority decision per I/O line, which can be implemented on-chip, as these have low pin count and run at comparably low clock frequencies. In our design we can utilize a combination of re-sampling majority voter and MUX as depicted in Figure 4. For packet-based interfaces such as Spacewire, AFDX, or CAN, no logic-side solution is necessary, as data duplication can be managed more efficiently at OSI layer 2+ [46].

We designed our MPSoC for FPGA due to the additional FDIR capability achievable through reconfiguration using different configuration variants. This allows our system to be self-healing by repairing defective tiles affected by upsets in tile logic, and to tolerate permanent faults using alternative configuration variants. However, an FPGA-based MPSoC implementation will not match the performance high-end mobile-market ASIC-SoCs clocked at 2GHz+, but it was also never designed to do so. OBCs of larger satellites today utilize processors clocked in the range of 10-300Mhz, whereas many CubeSats still utilize simple microcontroller SoCs. Our FPGA-based MPSoC can very well match the performance of these solutions, and depending on the utilized application is also capable of outperform them in direct comparison. Ignoring our design goal of achieving a cheap, adaptable OBC solution based on standard IP, a variant of this design could also be implemented on ASIC, allowing for drastically faster clock speeds at drastically increased cost.

VIII. Fault Injection Experiments

Radiation induced faults are the main challenge to our FT architecture. Hardware-side FT measures in general undergo radiation testing through exposure with a specific particle mix approximating the target environment. For the hardware used in our architecture, relevant radiation tests have been conducted already, e.g. in [8], [47], or are currently ongoing [37]. Tests for relevant memories are available in test databases such as ESCIES, NASA's NEPP[2] and the IEEE REDW Records.

Few related work regarding software-side FT measures has been subjected to actual, practical fault injection testing. Most academic publications on this subject does not consider actual implementations, while the required metrics are unavailable due to the use of classical fault-tolerance measures due to being based on proprietary hardware. Therefore, most related work relies on fault modeling using different statistical distributions as input for projections. However, SSEs and other particle induced faults are

[2]see https://escies.org and https://nepp.nasa.gov

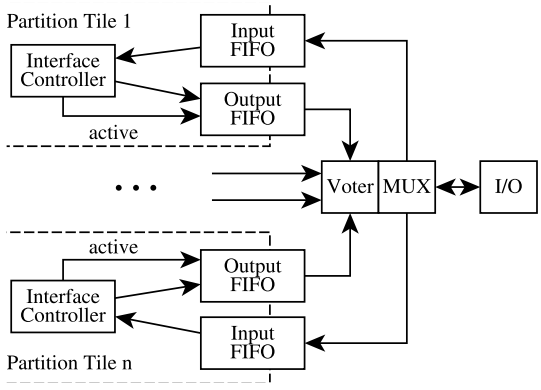

Fig. 4: Simplified representation of a activation-driven output voter input multiplexer for low-pin-count CubeSat interfaces. Note that an additional buffering/sampling step is required in case of different thread scheduling on lock-stepped tiles.

stochastic events, and for space applications an assumption about faults following a statistical distribution are non-representative. Hence, statistical models alone and approximate fault rates obtained from radiation-tests do not allow an assessment of our architecture's effectiveness.

Therefore, we conducted practical fault injection experiments to validate the effectiveness and efficiency of our lockstep approach in practice. In the remainder of this section, we provide preliminary results of our test campaign in comparison to [3], the only public, prior practical fault injection experiment conducted for a coarse-grain lockstep approach that we are aware of to date.

A. Experiment Setup

We conducted fault injection experiments on the coarse-grain lockstep approach using the open-source fault injection framework FIES [48], which was designed for systematically verifying the software-side functionality of COTS-based critical system. The FIES source code is available publicly[3] and licensed under GPLv2, in which we introduced several bug-fixes and adaptations, e.g. as original code required a generic simulation target. These patches will be made available at https://fieser.dependable.space.

Our coarse grained lockstep approach was implemented using the ARM Cortex-A/Zynq board-support-package of RTEMS 4.11.2, at that time the latest release. We compiled the kernel image with the RTEMS toolchain and standard flags (`-marm -mfpu=neon -mfloat-abi=hard -O2`).

As payload, we utilized *ESA's Next Generation DSP benchmark* run as POSIX threads on RTEMS, which is an ESA standard application used to compare DSP system performance. To re-confirm our results, we also tested using an application resembling the NASA/James Webb Space Telescope's Mid-Infrared Instrument's readout software [49], which we used in [2] for performance estimation.

We injected transient, intermittent, and permanent faults into memory, while transient and intermittent faults were injected into general purpose registers. Transient faults were injected as random bit-flips into the processor's state register (`CPSR`) and other special purpose registers and the processor pipeline. To do so, we generated memory traces using an extended version of the FIES framework's trace functionality, recording all read and write access of a tile. These measures allow us to model accurately radiation-induced faults. SEFIs in different functional units of a tile can induce faults which are neither permanent nor truly transient. To simulate SEFIs, we use FIES' intermittent fault functionality with a duration equivalent to 10 clock cycles at 100Mhz or 20 instructions executed by a Cortex-A pipeline.

B. Preliminary Results and Discussion

Table I depicts first results of our fault injection experiment. In payload-application code, a majority of the **transient faults** injected in memory and general purpose registers resulted in a corruption to the payload applications' state. With less than 20% of all faults, the application of the entire OS crashed or terminated prematurely (CPU resets were treated as early termination). As a majority of the payload application consists of data handling, memory access and arithmetic instructions, this is in line with our original expectations regarding our coarse grain lockstep implementation.

[3]https://github.com/ahoeller/fies.git

Impact	Recovery	Observed Effect per Type		
		Transient	Permanent	SEFI
Thread State	update	49%	44%	53%
Thread Crash	update	8%	17%	10%
Lockstep Fail	reboot	1%	2%	1%
OS Crash	reboot	10%	18%	15%
No Effect	-	32%	19%	21%

Tab. I: Preliminary fault injection experiment results. Notice that our setup does not enable us to detect test silent data corruption or faults resulting in incorrect I/O.

Faults affecting the lockstep mechanics themselves resulted in false comparison, incorrectly checksums generated from correct data or control flow deviations, but occurred seldom due to the minimal code and data footprint of the relevant functions. It is important to note that a sizable share of faults resulting in no noticeable effect, may also in practice be masked with no actual impact on the tile state [50].

The share of bit-flips resulting in a corrupted thread state due to **permanent faults** remained comparable as when injecting transients. However, this number by itself is misleading, as the amount of masked upsets without noticeable effects plummeted to just 19%, while thread- or OS-crashes increased. Therefore, we can deduct that a number of faults which due to transient faults would have resulted in just thread state corruption, now instead result in crashes. The total amount of detected faults in turn was increased by faults which were previously masked. **Intermittent faults** generated similar results to those with permanent faults, though with slightly fewer crashes and more faults affecting only the payload application.

Our results are consistent with related work [3], though the amount of faults resulting in application and OS crashes is larger. When conducting their transient fault injection experiments, Dobel et al. utilized highly application intrusive lockstep mechanics through function call hooking, which however offers more fine-grained fault-mitigation possibilities. In general, our campaign so far shows a higher amount of masked faults, a decreased amount of detected state deviations, and an increased amount of crashes with our approach. The authors of [3] consider their fault injection measurements overly optimistic, as they injected only transient faults into payload *"applications of little complexity (leading to few potential candidates for fault injection)* [3]", while assuming the OS and lockstep mechanics to be fault-free. In light of such bias, our reduced detection rates can be considered a practical consequence of testing with more complex applications running within a real OS.

This fault injection campaign is meant as intermediate step in validating our proof-of-concept implementation, and this paper contains early results only. A report on the full validation for our software-side FT concept is currently undergoing peer review [51]. Due to the positive outcome of the so far achieved test result, we are now developing a custom-PCB based prototype OBC. This prototype will then be used for radiation testing, to characterize its resilience and performance in a space-like environment.

IX. Conclusions

In this paper, we demonstrate the practical feasibility of the multi-stage fault-tolerance approach described in [2] through

constructing a prototype and evaluating its fault detection and recovery capabilities using practical fault injection into registers, memory and a tile's processors pipeline. Based on these results, we can deduce that our architecture does not exceed the tight energy, complexity and cost constraints of even very small spacecraft such as CubeSats with scientific or commercial payload, and is effective. To our knowledge, this is the first fault mitigation approach offering strong fault tolerance, which can uphold computational correctness viable for miniaturized spacecraft and is not dependent on proprietary processor cores.

We utilize fault tolerance measures across the embedded stack, and combine topological with software-side functionality to achieve the high level of reliability required to enable the use of nanosatellites in critical space missions. The resulting architecture enables an on-board computer to adapt to varying performance requirements at run-time, allowing processing capacity, energy consumption or fault coverage to be maximized. This design was implemented using only COTS hardware and widely available, pre-existing library IP, requiring no proprietary logic or costly space-grade processor cores. It was successfully tested on current generation Xilinx Zynq/Kintex and Virtex FPGAs with 4, 6 and 8 tiles, and validated using fault injection. We provide practical results for our architecture which was implemented as MPSoC design on a Xilinx XCKU5P FPGA with modest resource utilization.

Our current implementation serves as a proof-of-concept, and we conducted fault injection experiments using the FIES fault injection framework [48]. We provide preliminary results based on these experiments, which show that the fault detection and recovery capabilities of our approach are efficient and effective for protecting an FPGA-based on-board computer. A direct comparison of our results to literature demonstrates that our results are comparable to the protective strength of much tighter and more restrictive lockstep concepts. In contrast, our approach can be ported between operating systems and processor platforms rapidly, does not constrain the application structure or type, and is non-intrusive.

X. Acknowledgements

This approach was developed for a 4-year European Space Agency (ESA) NPI project, and we are implementing a prototype jointly with two industrial partners. We would like to thank Gianluca Furano, Giorgio Magistrati, Antonios Tavoularis and Kostas Marinis at ESTEC/TEC-EDD for their support and feedback. We also thank ARM Ltd. for providing access to processor and infrastructure IP. N.M. Murillo acknowledges funding through the European Union A-ERC grant 291141 CHEMPLAN, by the Netherlands Research School for Astronomy (NOVA), and the Royal Netherlands Academy of Arts and Sciences (KNAW).

References

[1] M. Langer and J. Bouwmeester, "Reliability of cubesats-statistical data, developers' beliefs and the way forward," in *AIAA SmallSat*, 2016.

[2] C. M. Fuchs *et al.*, "Bringing fault-tolerant gigahertz-computing to space," in *IEEE ATS*, 2017.

[3] B. Döbel, "Operating system support for redundant multithreading," Ph.D. dissertation, Dresden University, 2014.

[4] J. Schwank *et al.*, "Radiation Hardness Assurance Testing of Microelectronic Devices and Integrated Circuits," *IEEE Transactions on Nuclear Science*, 2013.

[5] M. Marinella and H. Barnaby, "Total ionizing dose and displacement damage effects in embedded memory technologies," Sandia National Laboratories, Tech. Rep., 2013.

[6] S. Bourdarie and M. Xapsos, "The Near-Earth Space Radiation Environment," *IEEE Transactions on Nuclear Science*, 2008.

[7] L. A. Tambara *et al.*, "Heavy ions induced single event upsets testing of the 28 nm xilinx zynq-7000 all programmable soc," in *Radiation Effects Data Workshop*. IEEE, 2015.

[8] M. D. Berg, K. A. LaBel, and J. Pellish, "Single event effects in FPGA devices 2014-2015," in *NASA NEPP/ETW*, 2015.

[9] M. Kochiyama *et al.*, "Radiation effects in silicon-on-insulator transistors with back-gate control method fabricated with OKI semiconductor 0.20 μm FD-SOI technology." Elsevier, 2011.

[10] F. Kastensmidt and P. Rech, *FPGAs and Parallel Architectures for Aerospace Applications: Soft Errors and Fault-Tolerant Design*. Springer, 2016.

[11] R. Carlson *et al.*, "On the use of system-on-chip technology in next-generation instruments avionics for space exploration," in *IEEE VLSI-SoC, revised paper*. Springer, 2016.

[12] M. Wirthlin, "High-reliability FPGA-based systems: space, high-energy physics, and beyond," *Proceedings of the IEEE*, vol. 103, no. 3, 2015.

[13] K. Reick *et al.*, "FT design of the IBM Power6 microprocessor," *IEEE micro*, 2008.

[14] M. Hijorth *et al.*, "GR740: Rad-hard quad-core LEON4FT system-on-chip," in *Eurospace DASIA*, 2015.

[15] A. S. Jackson, "Implementation of the configurable fault tolerant system experiment on NPSAT-1," Ph.D. dissertation, Naval Postgraduate School Monterey, 2016.

[16] X. Iturbe *et al.*, "On the use of system-on-chip technology in next-generation instruments avionics for space exploration," in *Springer VLSI-SoC*, 2015.

[17] S. Gupta *et al.*, "SHAKTI-F: A fault tolerant microprocessor architecture," in *IEEE ATS*, 2015.

[18] X. Iturbe *et al.*, "A triple core lock-step ARM Cortex-R5 processor for safety-critical and ultra-reliable applications," in *IEEE DSN*, 2016.

[19] R. DeCoursey *et al.*, "Non-radiation hardened microprocessors in space-based remote sensing systems," in *International Society for Optics and Photonics*, 2006.

[20] J. R. Samson, "Implementation of a dependable multiprocessor cubesat," in *IEEE Aerospace*, 2011.

[21] M. Pignol, "DMT and DT2," in *IEEE IOLTS*, 2006.

[22] L. Bozzoli and L. Sterpone, "Self rerouting of dynamically reconfigurable sram-based FPGAs," in *NASA/ESA AHS*. IEEE, 2017.

[23] M. Ebrahimi *et al.*, "Low-cost multiple bit upset correction in sram-based FPGA configuration frames," *IEEE Transactions on VLSI Systems*, 2016.

[24] F. Rittner *et al.*, "Automated test procedure to detect permanent faults inside sram-based FPGAs," in *NASA/ESA AHS*. IEEE, 2017.

[25] A. Stoddard, A. Gruwell, P. Zabriskie, and M. J. Wirthlin, "A hybrid approach to FPGA configuration scrubbing," *IEEE Transactions on Nuclear Science*, 2017.

[26] U. Kretzschmar *et al.*, "Synchronization of faulty processors in coarse-grained TMR protected partially reconfigurable FPGAs," *Elsevier RESS*, 2016.

[27] A. Shye *et al.*, "Using process-level redundancy to exploit multiple cores for transient fault tolerance," in *IEEE DSN*, 2007.

[28] J. Hursey *et al.*, "The design and implementation of checkpoint/restart process fault tolerance for open MPI," in *IEEE IPDPS*, 2007.

[29] P. Munk *et al.*, "Toward a fault-tolerance framework for COTS many-core systems," in *IEEE EDCC*, 2015.

[30] L. Zeng, P. Huang, and L. Thiele, "Towards the design of fault-tolerant mixed-criticality systems on multicores," in *ACM CASES*, 2016.

[31] S. P. Azad *et al.*, "Holistic approach for fault-tolerant network-on-chip based many-core systems," *ACM HiPEAC, DREAMCloud*, 2016.

[32] A. Höller *et al.*, "Software-based fault recovery via adaptive diversity for COTS multi-core processors," 2015, arXiv:1511.03528.

[33] A. D. Santangelo, "An open source space hypervisor for small satellites," in *AIAA SPACE*, 2013.

[34] E. Missimer, R. West, and Y. Li, "Distributed real-time fault tolerance on a virtualized multi-core system," *Euromicro ECRTS, OSPERT*, 2014.

[35] Z. Al-bayati *et al.*, "Fault-tolerant scheduling of multicore mixed-criticality systems under permanent failures," in *IEEE DFT*, 2016.

[36] A. K. Singh, M. Shafique, A. Kumar, and J. Henkel, "Mapping on multi/many-core systems: survey of current and emerging trends," in *DAC*. ACM, 2013.

[37] P. Meloni *et al.*, "System adaptivity and fault-tolerance in NoC-based MPSoCs: the MADNESS project approach," in *IEEE DSD*, 2012.

[38] N. K. R. Beechu *et al.*, "Hardware implementation of fault tolerance NoC core mapping," *Springer Telecommunication Systems*, 2017.

[39] C. M. Fuchs *et al.*, "Dynamic fault tolerance through resource pooling," in *NASA/ESA AHS*. IEEE, 2018.

[40] C. M. Fuchs, C. Trinitis, N. Appel, and M. Langer, "A fault-tolerant radiation-robust mass storage concept for highly scaled flash memory."

[41] S. S. Arnold, R. Nuzzaci, and A. Gordon-Ross, "Energy budgeting for cubesats with an integrated FPGA," in *IEEE Aerospace Conference*, 2012.

[42] A. Cristo *et al.*, "Optimization of processor-to-hardware module communications on spaceborne hybrid FPGA-based architectures," *IEEE Embedded Systems Letters*, vol. 5, no. 4, pp. 77–80, 2013.

[43] C. M. Fuchs *et al.*, "Preliminary Performance Estimations and Benchmark Results for a Software-based Fault-Tolerance Approach aboard Miniaturized Satellite Computers," 2017, arXiv:1706.02086.

[44] Z. Zhang *et al.*, "Single event effects in COTS ferroelectric RAM technologies," in *Radiation Effects Data Workshop*. IEEE, 2015.

[45] G. Tsiligiannis *et al.*, "Testing a commercial MRAM under neutron and alpha radiation in dynamic mode," *IEEE Transactions on Nuclear Science*, 2013.

[46] Aeronautical Radio, INC, *ARINC Specification 664: Avionics Full Duplex Switched Ethernet (AFDX)*, 2005.

[47] D. S. Lee *et al.*, "Single-event characterization of the 20 nm xilinx kintex ultrascale field-programmable gate array under heavy ion irradiation," in *Radiation Effects Data Workshop*. IEEE, 2015.

[48] 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.

[49] M. Ressler *et al.*, "The mid-infrared instrument for the James Webb Space Telescope," *Astronomical Society of the Pacific*, 2015.

[50] X. Li and D. Yeung, "Application-level correctness and its impact on fault tolerance," in *HPCA*. IEEE, 2007.

[51] C. M. Fuchs *et al.*, "How to validate software-side fault tolerance measures for spaceflight by example." submitted, 2018.