A Fault-Tolerant MPSoC For CubeSats

Christian M. Fuchs*§, Pai Chou§, Xiaoqing Wen¶, Nadia M. Murillo†,
Gianluca Furano‡, Stefan Holst¶, Antonios Tavoularis‡, Shyue-Kung Lu‖, Aske Plaat*, and Kostas Marinis‡
*Leiden Observatory, ‡European Space Agency, Netherlands, ¶Kyushu Institute of Technology, Japan
§National Tsing Hua University ‖National Taiwan University of Science and Technology, Taiwan
email: christian.fuchs@dependable.space

***Abstract*—We present the implementation of a fault-tolerant MPSoC for very small satellites (<100kg) based upon commercial components and library IP. This MPSoC is the result of a co-design process and is designed as an ideal platform for software-implemented fault-tolerance measures. It enforces strong isolation between processors, and combines fault-tolerance measures across the embedded stack within an FPGA. This allows us to assure robustness for a satellite on-board computer consisting of modern semiconductors manufactured in fine technology nodes, for which traditional fault-tolerance concepts are ineffective. We successfully implemented this design on several Xilinx UltraScale and UltraScale+ FPGAs with modest utilization. We show that a 4-core implementation is possible with just 1.93 W of total power consumption, which for the first time enables true fault-tolerance for very small spacecraft such as CubeSats. For critical space missions aboard heavier satellites, we implemented an MPSoC-variant for the space-grade XQRKU060 part together with the Xilinx Radiation Testing Consortium. The MPSoC was developed for a 4-year ESA project. It can satisfy the high performance requirements of future scientific and commercial space missions at low cost while offering the strong fault-coverage necessary for platform control for missions with a long duration.**

I. Introduction

Satellite miniaturization has enabled a wide variety of scientific and commercial space missions, which previously were technically infeasible, impractical, or simply uneconomical. However, due to their low reliability, miniaturized satellites (<100kg) are typically considered unsuitable for critical space missions and high-priority science. The on-board computer (OBC) and related electronics, which constitute a large part of such spacecraft, have been shown responsible for a significant share of post-deployment failure [1]. Indeed, these components often lack even basic fault tolerance (FT) capabilities.

Due to budget, energy, mass, and volume restrictions, existing FT solutions originally developed for larger spacecraft cannot be adopted. In this paper, we present the implementation of an MPSoC designed specifically as platform for a hardware-software hybrid fault-tolerance architecture. We utilize topological features to achieve a design that allows software-implemented FT concepts to maximize their fault-coverage strength, providing FT even for CubeSats. The MPSoC can be assembled from well-tested COTS components, library IP, and powerful mobile-market processor cores. We have successfully implemented it on several different FPGA families.

In the next section, we introduce the fault-profile of our application and discuss related work. Subsequently, we describe our MPSoC and our cross-stack fault-tolerance approach. We provide implementation results in Section IV. Finally, we discuss advanced applications of our proof-of-concept, and conclude this paper with a comparison to miniaturized and large satellite computing.

II. Background and Related Work

The main sources of faults in the space environment are highly charged particles. These are ejected by the Sun, especially during Solar Particle Events, and arrive as Cosmic Rays from beyond our solar system [2]. Particles interact with a spacecraft's electronics, and can induce different effects depending on the type of particle and its charge. They can corrupt logical operations and induce bit-flips within semiconductor logic and memory (single event effects - SEE), and may cause displacement damage (DD) at the molecular level. The cumulative effect of charge trapping in the oxide of electronic devices (total ionizing dose – TID) further impacts the lifetime of an OBC. Radiation events can also cause functional interrupt in circuits, interfaces, or even entire chips (single event functional interrupts – SEFIs). The energy threshold above which a particle can induce transient faults in chips manufactured in finer technology nodes decreases. In turn, the ratio of multi-bit upsets or permanent faults increases. However, recent-generation semiconductor manufactured in certain modern technology nodes (e.g. FinFET and FD-SoI) show better performance under radiation than predicted by models [3].

A. Fault-Tolerance for Spacecraft

Traditional FT computers for spaceflight [4] implement circuit- or RTL- [5], IP-block- [6], [7], and OBC-level TMR [8] using space-proprietary IP. Circuit-, RTL-, and core-level measures are effective for small microcontroller-SoCs [9], [10], if they are manufactured in large feature-size technology nodes. Additional FT-circuitry is needed to compensate for the increased severity of radiation effects with modern technology nodes [9]. Processor lockstep implemented in hardware lacks flexibility, limits scalability, and is feasible only for very small MPSoCs with few cores [10], [11]. Timing and logic placement becomes increasingly difficult for more sophisticated processor designs, and becomes infeasible for SoCs running at higher clock frequencies. Practical applications run at very low clock frequencies with two or three simple processor cores, even for ASIC implementations [6], [10].

Nanosatellite developers instead use the same energy efficient, cheap modern electronics [12] for which traditional radiation-hardening concepts become ineffective. Especially CubeSats utilize COTS microcontrollers and application processor SoCs, FPGAs, and combinations thereof [12], [13]. Some computer designs for nanosatellites utilized redundancy at the component level to achieve fail-over, and provide at least some protection from failure. However, practical flight results show that such designs are complex and fragile, as compared to entirely unprotected ones [12], [14]. These in turn may fail at any given point in time, but CubeSat designers today are forced to accept this risk and hope that their system will not experience critical faults before their satellite's mission is concluded. Risk acceptance is viable only for brief educational, uncritical, low-priority missions.

B. Fault-Tolerance Concepts for COTS Technology

Commodity FPGAs have become popular for miniaturized satellite applications as they allow a reduction of custom logic and component complexity. FPGA-based SoCs can offer increased FDIR potential in space over ASICs manufactured in the same technology nodes [13]. Transients in configuration memory (CRAM) can be recovered through reconfiguration [15]. However, fine-grained, non-invasive fault detection in FPGA fabric is challenging [9], and is a subject of ongoing research [16], [17]. Applications thus rely on scrubbing, which has scalability limitations and covers only parts of the fabric.

Software-implemented fault-tolerance concepts for multi-core systems were identified as promising already in the early days of microcomputers [18], but they have been technically infeasible and inefficient until few years ago. Modern semiconductor technology allows us to overcome these limitations, and recent research

978-1-7281-2260-1/19/$31.00

[19], [20] has shown that modern MultiCore-MPSoC architectures can theoretically be exploited to achieve FT. However, current concepts are incapable of general-purpose computing, and instead cover deeply embedded applications with a very specific software structure [21], [22]. They require custom processor designs [19], or programming models that are suitable for accelerator applications [20]. They are not meant to run a full operating system with standard software. The fundamental concept of software-implemented coarse-grain lockstep, however, is flexible and can be applied, e.g., to MPSoCs for safety-critical applications [19], [23], networked, distributed, and virtualized systems [24].

III. Fault-Tolerance Through Co-Design

Software-implemented fault-tolerance concepts only offer weak protection, unless combined with other protective measures to achieve synergies [23]. To assure sufficient long-term protection for space applications in space missions with an extended duration with just COTS component, a suitable architecture and system topology is required. In this section, we describe how a hardware-software co-designed MPSoC can achieve strong fault-tolerance by combining software-implemented fault-tolerance measures with topological features, error correction coding (ECC), and FPGA reconfiguration. Not only can this MPSoC run unmodified standard application software but also efficiently counter resource exhaustion and age gracefully. It consists of only COTS technology and requires no proprietary FT processor IP, hardware-lockstep, or other scalability-limiting design measures.

A. Software and System-Level Fault-Tolerance

The main target platform for our research is the ARM Cortex-A53 application processor, which is widely used in mobile-market devices. However, our system architecture is independent of the processor and platform. For low-cost CubeSats, the use of proprietary IP will often be impractical due to cost constraints and the time necessary to obtain such IP. Hence, freely available soft-cores such as the mature MicroBlaze by Xilinx are better suited for very small satellite applications. The MPSoC described in this paper therefore uses MicroBlaze soft-cores, and provides an ideal platform for this multi-stage fault-tolerance approach:

Stage 1 utilizes **coarse-grain lockstep** to generate a distributed majority decision between threads replicas and implements **forward error correction**. It maintains **state-synchronization between applications** running on a set of otherwise independent processor cores. We analyzed the computational cost of our lockstep in [26] and validated it through fault-injection in [27]. Stage 1 utilizes thread-migration, and requires spare processing capacity in the system to handle permanent faults.

Stage 2 corrects faults in an **FPGA's CRAM** and **recovers** the integrity of **processor cores** to counteract exhaustion of spares and processing capacity. Its objective is to repair partial-reconfiguration partitions through reconfiguration and the use of differently routed and placed partition variants.

Stage 3, described in detail in [28], engages when then system has **exhausted all available spares** due to accumulating permanent defects in an aged FPGA. It **re-allocates processing time** between applications to maintain a reliable system. We utilize the **mixed criticality** properties inherent to on-board data processing, assuring **sufficient compute resources** are available to **high-criticality applications**. This enables an MPSoC to **age gracefully** by trading performance for energy saving or robustness.

B. Achieving Robustness through Topology

An MPSoC designed for robustness must utilize a topology that avoids single points of failure and supports FDIR where possible. However, conventional MPSoCs follow a centralist topology where cores share infrastructure as much as possible to minimize the design's footprint, optimize timing, improve routing, thereby achieving higher clock frequencies [29]. Therefore, conventional

Fig. 1: Logic placement of our proof-of-concept quad-core MPSoC for the upcoming XRTC Kintex UltraScale KU60 device-test board. A QSPI controller is highlighted in teal for size-comparison between an interface core and a compartment's total size.

MPSoC typologies actively try to reduce isolation between cores to make inter-process-communication and thread-migration less costly. Considering the impact of faults on an application's control flow and data path, this implies little to no isolation for software running on different processor cores. Faults in one core may therefore induce negative effects into other cores. From a fault-tolerance perspective, this is undesirable, and we instead implement an MPSoC topology that offers isolation by design. Its logic placement is depicted in Figure 1, and we will discuss the rationale behind our architecture subsequently.

To keep application replicas isolated, we place each processor core within a separate *compartment*. A compartment is comprised of the minimum set of IP-blocks constituting a conventional single-core SoC, including interrupt controller, peripheral controllers, I/O, and bring-up software. Each compartment is outfitted with an interconnect access bridge to allow low-level diagnostics and access to the compartment's address space by an off-chip supervisor. A compartment runs an instance of the OS, is assigned one or multiple applications, has access to hardware timers, interrupts, and can handle peripheral I/O autonomously. Besides an on-chip memory holding the bootloader, it is also outfitted with a dedicated dual-port state-memory used to exchange lockstep information. A block diagram is depicted in Figure 2.

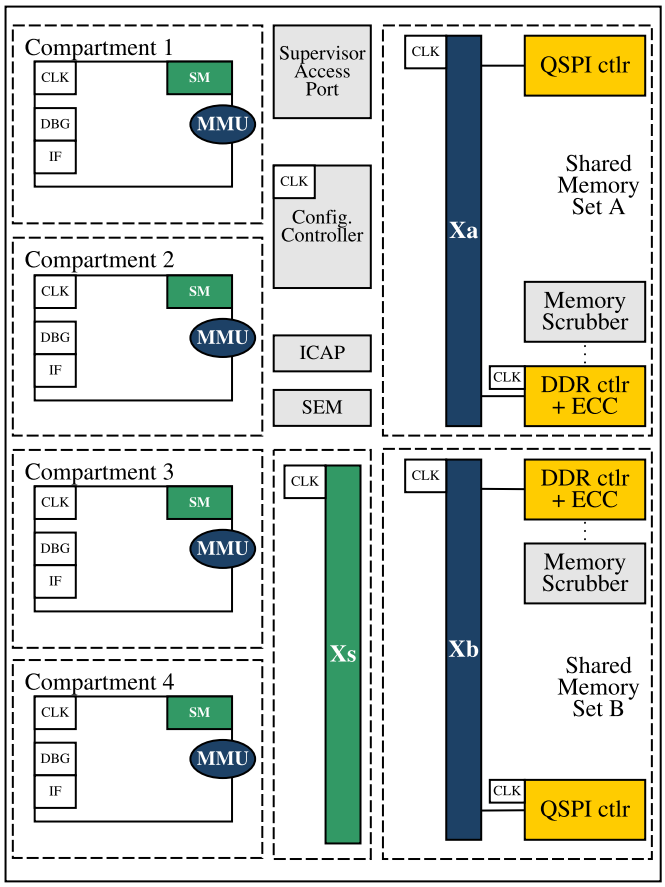
On-chip memory is insufficient to store a modern operating system and application software. Hence, a compartment also requires access to larger non-volatile mass-memory and DDR/SDRAM. PCB layout constraints, energy constraints, complexity, and the large footprint of DDR memory controllers make it infeasible to instantiate these memories for each compartment. Therefore, we implement a two-tiered interconnect architecture as depicted in Figure 2, with off-chip memory and their controllers being shared between compartments in sets.

The main source of faults in commercial memory ICs are data corruption due to SEE-induced bit-upsets and SEFIs [30]. We counter bit-upsets through the use of ECC and interconnect-level error scrubbing. In our MPSoC, we realize SECDED coding using library and open-source IP, which are usually considered sufficient for CubeSat-use. Stronger protection can be achieved using commercial IP. To avoid introducing a single point of failure and to handle SEFIs economically, we implement redundant memories controller sets. In case one set fails and requires reconfiguration, the compartments can switch over to the fail-over instance at runtime. The availability of multiple controller sets also enables load-distribution for memory access, and reduces latency due to concurrent access by multiple compartments. In practice, compartments thus cannot saturate DDR controllers.

The address space of all compartments is uniform, enabling memory structures to be migrated between compartments and re-used. To each compartment and in each controller set, we allocate a main memory segment. A compartment has read-only access to the other segments. Segments are isolated through the interconnects topology, which requires modifying the interconnect IP's configuration in order to enable write access. Through the MMU component indicated in Figures 2 and 3, we perform the necessary address translation operations. A failed compartment therefore cannot corrupt another compartment's memory segment. This eliminates a system-wide single point of failure.

To efficiently perform lockstep state comparison and synchronization between compartments, an MPSoC has to provide adequate means of exchanging state-data. For small MPSoCs with less than six cores, this is realized in DDR/SDRAM memory. For larger designs, a dedicated state-exchange network improves performance and offers stronger isolation. These components are depicted in green in the figures. Access to state memory then takes place entirely on-chip without passing through caches and the global interconnect.

Fig. 3: Partition layout and clocks in our MPSoC. Partitions are indicated with dashed lines. Compartment and memory controller sets ($X_{a/b}$) can be reconfigured without interruption.



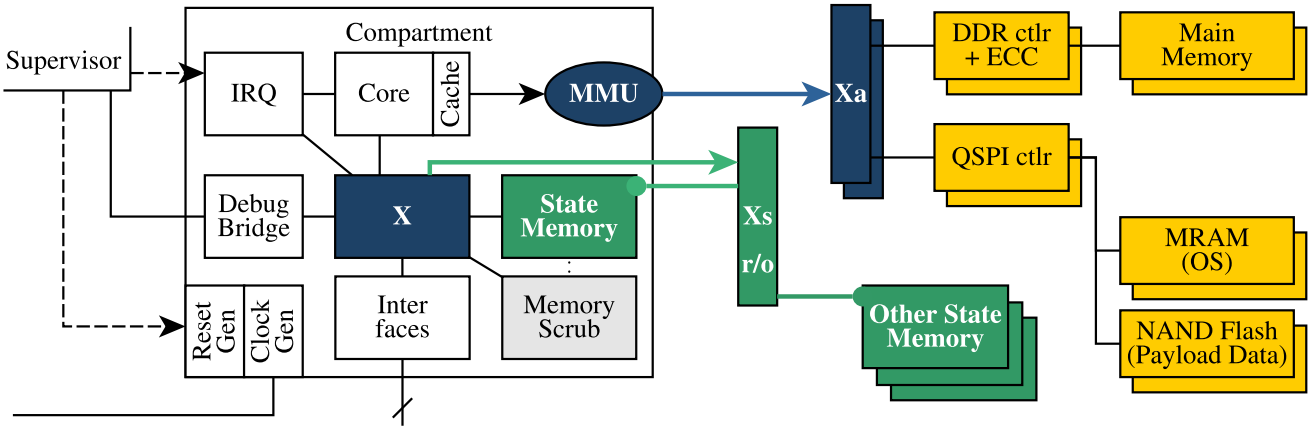


Fig. 2: The high-level architecture of a compartment. Access to local IP and state memory bypasses the cache, while access to global memory is cached to avoid interconnect congestion. MRAM memory cells are inherently radiation immune [25].

C. Logic Isolation and Partitioning

Permanent damage to FPGA-fabric can be mitigated though reconfiguration with a differently routed and placed configuration variant [15]. However, full reconfiguration interrupts the operation of the implemented MPSoC. To enable continued operation during reconfiguration, each compartment and memory controller set resides in a dedicated partial reconfiguration partition. Partial reconfiguration also increases the likelihood that a suitable combination of partition-variants can be found to mitigate all permanent faults present in the FPGA's fabric.

Large clock trees and reset networks are known to be problematic in space applications [31]. The logic in each compartment resides in a separate clock domain, and a memory controller set controls each of the DDR4 backend, memory controller front-ends, and AXI-interconnects. Therefore, the clock trees are isolated from each other and are de-coupled on the AXI interconnects of the memory controller sets. This minimizes clock skew and its impact, as well as temperature-related effects, while improving timing and logic routing.

To protect the running configuration of our SRAM-based FPGA, we implement CRAM-frame ECC using the Xilinx Soft Error Mitigation IP (SEM [32]). However, configuration-level erasure coding and scrubbing can still only detect faults in specific components of the FPGA fabric (e.g., not in BlockRAM). We address this limitation at the system level: the coarse-grained lockstep functionality in Section III-A enables us to detect faults in the fabric with compartment granularity within 1-3 lockstep cycles [26]. In practice, this closes the fault-detection gap left by scrubbing and configuration erasure coding. This process is described in detail in [27].

We place a reconfiguration controller in static logic to perform high-speed reconfiguration via ICAP. It communicates with the Xilinx UltraSEM IP and can be deactivated by the supervisor in case of failure, in which case the off-chip supervisor can also deactivate via JTAG. It has access to the FPGA's configuration memory through an SPI controller core. Architecturally, we implement the configuration controller using a stripped-down compartment design without the capability to run a full OS.

D. External Components

In our proof-of-concept, we utilized DDR4-DRAM as main memory, Magnetoresistive RAM (MRAM) [25] to store operating system code and data, NAND-flash as mass memory for application data, and NOR-Flash to store the FPGA's configuration. DRAM and NOR-flash are commonly used in all FPGA-based systems and their use currently cannot be avoided. However, relatively low-cost radiation-robust variants of these components are available commercially. Commercial MRAM memory cells are inherently radiation immune [25] and are today widely used aboard CubeSats due to their excellent performance in the space environment. To store payload data, radiation-soft NAND-Flash is used. At the time of writing, flash memory is widely used in all space applications, and currently there is no readily available radiation-immune alternative. We mitigate SEFIs and faults in addressing and control logic through the use of software-level error detection and correction at the file-system and block level [27]. Radiation-immune phase-change memory [33] can be a prime candidate for replacing NAND flash once commercially available.

The off-chip supervisor above is implemented through a low-performance microcontroller and performs management tasks as well as debugging. It triggers checkpoints and receives voting results from the compartments via a set of GPIO pins and maintains a fault-counter for each compartment. In our proof-of-concept MPSoC, it also adjusts thread-assignment for the coarse-grain lockstep, but this is a simplification to reduce development time. The supervisor takes no part in the normal data processing operations of the OBC and requires very little processing power. Therefore, a variety of available radiation-robust low-performance products can also be used aboard CubeSats.

IV. Utilization and Power Comparison

The quad-core MPSoC architecture described in this paper has been implemented on a set of Kintex UltraScale and UltraScale+ devices using Xilinx MicroBlaze soft-cores running at 300 MHz, and DDR4 controllers. In our proof-of-concept, we utilize a FeRAM-based MSP430FR5969 controller for our proof-of-concept, for which a low-cost space-grade substitute is available. The MPSoC is reproducible in Xilinx Vivado 2017.1 and later. The necessary IP is included in the Vivado IP library and can be obtained free of charge through Xilinx's university program by academics and non-commercial scientific users. This serves as proof-of-concept for our architecture, with resource utilization indicated in Table I.

For this MicroBlaze-based MPSoC implementation, the added logic footprint for instantiating a compartment is low compared to just an application-processor without any peripherals. For size comparison between an interface IP-core and a compartment, a QSPI controller core is highlighted in Figure 1 in teal. It makes up only 2.5% of a compartment's LUT and 6% BRAM utilization, with other commonly used cores aboard CubeSat such as I2C or UART showing a similar or even lower footprint. The larger size of ARM Cortex-A53 processor cores reduces this ratio even further.

Our initial proof-of-concept was implemented on the Xilinx Virtex UltraScale+ VCU118 Evaluation Kit with DDR4 controllers running at 1600 MHz. This FPGA family was ideal for design space exploration as the kit has two DDR4 memory channels and a large fabric. Within the Xilinx Radiation Test Consortium, we have been porting our design to the consortium's upcoming Kintex UltraScale KU60/XQRKU060 test board for radiation testing. Logic and partition placement are depicted in Figure 1. Tables I and II show the FPGA utilization and power consumption. On KU60, DDR4 memory controllers run at 1000 MHz due to generational constraints.

We ported our MPSoC also to smaller Kintex UltraScale+ devices, including the KU60's closest equivalent part KU11P and the smallest FPGA in the family and generation, the KU3P. The port required minor adjustments to the utilization of clocking resources, as both KU11P and KU3P have fewer clocking-resource (MMCM and PLL tiles) than the KU60. On KU11P, it was sufficient to switch several clock-generators used in the shared memory controller sets from PLL to MMCM tiles, without changing other parameters. The main constraint of the KU3P, however, required a reduction of clock generators in memory controller sets to 1 clock domain instead of 3 as described in Section III-C. Due to the much smaller fabric of the KU3P, clock-domain sizes and routing distances decrease, resulting in better timing of the design.

Despite much lower dynamic power consumption across the board in UltraScale+, the KU11P variant shows slightly higher static power consumption than the KU60, which is counter-intuitive. After discussion within the Xilinx Radiation Testing

	KCU3P %	KCU11P %	KCU60 %
LUT	52.55%	29.20%	39.91%
LUTRAM	9.33%	6.49%	13.30%
FF	28.81%	16.08%	23.91%
BRAM	**84.31%**	50.58%	29.26%
DSP	2.19%	1.02%	1.09%
IO	73.68%	43.75%	60.58%
BUFG	8.20%	3.20%	4.17%
MMCM	**50.00%**	**25.00%**	16.67%
PLL	**87.50%**	**56.25%**	54.17%

TABLE I: Resource utilization our MPSoC on different Xilinx Kintex FPGAs. The XRTC variant's DDR4 memory controllers has a larger data-width due to package constraints.

TABLE II: Power consumption of the 3 MPSoC implementations. Data from the Vivado Implementation Power Report.

	KCU3P	KCU11P	KCU60
Clocks	0.22W	0.29W	0.71W
Signals	0.11W	0.15W	0.30W
Logic	0.11W	0.15W	0.42W
BRAM	0.19W	0.19W	0.41W
DSP	<0.01W	<0.01W	<0.01W
PLL	0.37W	0.46W	0.72W
MMCM	0.23W	0.23W	0.21W
I/O	0.27W	0.34W	1.50W
Dynamic	**1.51W**	**1.81W**	**4.26W**
Static	**0.43W**	**0.70W**	**0.67W**
Total Power	**1.94W**	**2.51W**	**4.93W**

Consortium, the most plausible explanation for this anomaly is the different IO-bank placement within the fabric between these devices. On KU60, IO-banks are placed in more favorable locations considering MPSoC design than on KU11P. This increases logic-spread, leaving less fully inactive fabric sections, which could explain an increase in static power consumption due to infrastructure on KU60.

The resulting UltraScale+ MPSoC implementations, while functionally equivalent, show a 50% lower power consumption than the previous generation. This is due to manufacturing in a 16 nm FinFET technology node instead of 20 nm planar. Power savings mainly come from a reduced dynamic power consumption of this design, due to an increased degree of logic concentration in a smaller FPGA-fabric area. For CubeSat-use, the Kintex UltraScale+ family is therefore more attractive, despite the acceptable potential risk of IO-pin latch-up [34], which today is mitigated in related work [35] at the system level. On the the smallest UltraScale+ part `xcku3p-sfvb784` available at the time of writing, we achieved 1.94 W total power consumption. This is well within the power budget range of existing 2U CubeSats, as shown through comparison in Table III.

Synthesis was run in "Alternative Routability", and implementation in "Performance-Explore" strategy with post-route placement & power optimization, as the resulting implementations showed consistently better timing and power utilization.

V. Experimental Results and Validation

We have tested this MPSoC on Xilinx VCU118 with two DDR memory channels, and on a KCU116 board with one channel (due to board constraints). From these setups, we have successfully developed a breadboard OBC-setup with an MSP430FR MCU.

In 2018, we tested our architecture through fault injection using system emulation, and published preliminary test results in [27]. In 2019, we constructed a multi-core model of our MPSoC also in ArchC/SystemC on RISC-V to compare and reconfirm our tests from 2018 with a different fault-injection technique closer to hardware. These results show that with near statistical certainty, a fault affecting a compartment can be detected within 1–3 lockstep cycles. This demonstrates that Stage 1 is effective and works efficiently. A full report on this validation is pending publication: it includes results for a 3-way comparison with different fault-injection techniques, plus upcoming test results for another lockstep implementation published by Dobel et al. in [36]. Next, we plan to construct a prototype for radiation testing with the Xilinx Radiation Testing Consortium and eventually for on-orbit demonstration in a CubeSat.

VI. Handling FPGA Failure and SEFIs

Our proof-of-concept MPSoC design spans only of a single FPGA and is not designed to withstand component-wide SEFIs affecting the entire FPGA. However, it can be implemented to tolerate such faults and even full component failure. The system depicted in Figure 4b implements our architecture on two FPGAs and does not suffer these limitations: instead of implementing all compartments and shared memory controller sets on a single FPGA, they can be distributed across multiple FPGAs. The failure of, e.g., a memory component or connected to one FPGA or its controller, does not cause the failure of an entire redundant system side. Compartments on one FPGA connected to a failed component can access components on the B-side. Even a severely degraded system implementing our architecture that has suffered multiple component failures can thus still operate correctly and support non-stop operation. In contrast to a traditional OBC based on component-redundancy, our architecture thus can deliver stronger fault-tolerance capabilities than traditional OBCs.

To support larger MPSoCs with more than eight compartments efficiently, a more scalable interface between compartments and memory controller sets should be used, such as a Network-on-Chip (NoC). An NoC not only allows drastically larger MPSoC designs [37] due to improved scalability, but also enables fault-tolerant routing [20], backwards error correction (re-transmission), and quality-of-service support [38]. When implementing our architecture with an NoC, the shared memory controller sets would be implemented as one NoC layer, while the state-exchange network forms a second layer. In contrast to conventional interconnects topologies, NoC routers can also implement error correction [39].

VII. Conclusions and Future Work

We have presented the implementation of an MPSoC that can assure robustness for a satellite on-board computer consisting of modern semiconductors manufactured in modern technology nodes. It is the result of a hardware-software co-design process and utilizes fault-tolerance measures across the embedded stack. We utilize a set of software-implemented fault-tolerance measures and realize forward-error-correction through coarse-grain lockstep. This functionality is combined with erasure coding, error

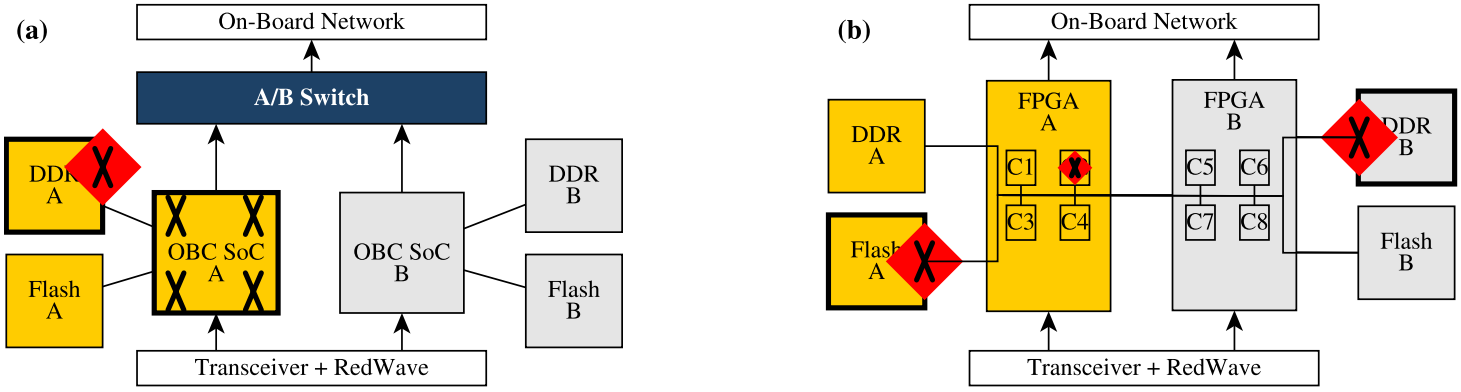


Fig. 4: **(a):** A traditional redundant system where there A-side failed due to malfunction in one memory components, which will fail once a fault occurs on the B side. **(b):** a system implementing our architecture, which is still functional and not degraded, even though multiple components have failed on both sides.

TABLE III: Comparison of the proposed MPSoC to 5 existing solutions for large spacecraft (grey) and CubeSats. Manufacturer data sheets information current as of July 2019. Clydespace OBC p.n. 01-02928. GR740 power value from Gaisler's Power Calculator.

	RAD5545	GR740	NanoMind Z7000	**Our MPSoC**	ClydeSpace	CubeSatKit D2
Architecture	MPSoC	MPSoC	MPSoC	**MPSoC**	SoC	uC
Implementation	ASIC	ASIC	Hybrid	**FPGA**	FPGA	ASIC
# Cores	4, 5600MIPS total	4, @250Mhz	2, @800Mhz	**4+, @300Mhz+**	1, @50Mhz	1, <80Mhz
CPU Core	RAD5500	LEON4	Cortex-A9	**MicroBlaze**	Cortex-M3	dsPIC33
RAM+OCM	4GB	4GB max.	1GB	**4GB+**	8MB	286KB
Storage	1GB	external	32GB	**external**	4GB	8MB
Free Toolchain	No + NDA	No + NDA	yes	**yes**	yes	yes
Software	Linux & Co	Linux & Co	Linux & Co	**Linux & Co**	RTOS	bare metal
Cost	high	high	low	**low**	low	lowest
Max Power	>20W	1.65W	2.3W	**1.94W**	1W	0.73W
Fault-Tolerant	yes	yes	no	**yes**	no	no

scrubbing, partial FPGA reconfiguration, and an MPSoC topology designed for logic and data isolation.

A comparison to existing traditional space-grade solutions as well as those available to CubeSat developers seems unfair, but is depicted in Table III. Today, miniaturized satellite computing can use only low-performance microcontrollers and unreliable MPSoC designs without fault-tolerance capabilities. Using the same type of commercial technology, our MPSoC can assure long-term fault coverage through a multi-stage fault-tolerance architecture, without requiring fragile and complex component-level replication. Considering the few more robust, low-performance CubeSat compatible microcontrollers for which the PIC-equipped CubeSatKit D2 OBC is representative, our implementation can offer a beyond factor-of-10 performance improvement even today. Our current academic proof-of-concept implemented on FPGA exceeds the single-core performance of the latest generation of space-grade SoC-ASICS such as an GR740, while offering fault-tolerance capabilities. We do so at a fraction of the cost and without the tight technological constraints of traditional space-grade technology.

Traditional fault-tolerant computer architectures intended for space applications struggle against technology and are ineffective for embedded and mobile-market components. Instead, we designed a software-based fault-tolerance architecture and this MPSoC specifically to enable the use of commercial modern semiconductors in space applications. We do not require any space-grade components, fault-tolerant processor designs, other custom, or proprietary logic. Our proof-of-concept is designed for ARM Cortex-A53 cores widely used in COTS MPSoCs, while the reproducible design variant described in this contribution utilizes MicroBlaze. It can be replicated with just standard design tools and library IP, which are available free of charge to many designers in academic and research organizations. Therefore, our architecture scales with technology, instead of struggling against it. It benefits from performance and energy efficiency improvements that can be achieved through improved technology nodes, and scales for designs with more, and powerful processor cores.

ACKNOWLEDGMENTS

We would like to thank Giorgio Magistrati of ESA/ESTEC, Kozo Takeuchi of JAXA, our colleagues Gary Swift and Sebastian E. Garcia of the Xilinx Radiation Test Consortium, and Melanie Berg of Space R2 LLC and and NASA Goddard Space Flight Center for their valuable feedback, discussions, support and encouragement. We thank ARM Ltd. for making available the relevant processor and infrastructure IP.

REFERENCES

[1] M. Langer and J. Bouwmeester, "Reliability of cubesats-statistical data, developers' beliefs and the way forward," in *AIAA SmallSat*, 2016.
[2] J. Schwank *et al.*, "Radiation Hardness Assurance Testing of Microelectronic Devices and Integrated Circuits," *IEEE Transactions on Nuclear Science*, 2013.
[3] M. D. Berg, K. A. LaBel, and J. Pellish, "Single event effects in FPGA devices 2014-2015," in *NASA NEPP/ETW*, 2015.
[4] G. Lentaris *et al.*, "High-performance embedded computing in space: Evaluation of platforms for vision-based navigation," *Journal of Aerospace Information Systems*, 2018.
[5] C. Carmichael, "Triple module redundancy design techniques for Virtex FPGAs," *Xilinx Application Note XAPP197*, 2001.
[6] K. Reick *et al.*, "FT design of the IBM Power6 microprocessor," *IEEE micro*, 2008.
[7] M. Hijorth *et al.*, "GR740: Rad-hard quad-core LEON4FT system-on-chip," in *Eurospace DASIA*, 2015.
[8] K. D. Safford *et al.*, "Off-chip lockstep checking," Jun. 26 2007, uS Patent 7,237,144.
[9] A. Fedi *et al.*, "High-energy neutrons characterization of a safety critical computing system," in *IEEE DFT*. IEEE, 2017.
[10] X. Iturbe *et al.*, "A triple core lock-step ARM Cortex-R5 processor for safety-critical and ultra-reliable applications," in *IEEE DSN-W*, 2016.
[11] Á. B. o. de Oliveira, "Applying lockstep in dual-core arm cortex-a9 to mitigate radiation-induced soft errors," in *LASCAS*. IEEE, 2017.
[12] M. Swartwout, "The first one hundred CubeSats: A statistical look," *Journal of Small Satellites*, 2014.
[13] 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.
[14] J. Bouwmeester, M. Langer, and E. Gill, "Survey on the implementation and reliability of cubesat electrical bus interfaces," *CEAS Space Journal*, 2017.
[15] L. Bozzoli and L. Sterpone, "Self rerouting of dynamically reconfigurable SRAM-based FPGAs," in *NASA/ESA AHS*. IEEE, 2017.
[16] M. Ebrahimi *et al.*, "Low-cost multiple bit upset correction in SRAM-based FPGA configuration frames," *IEEE Transactions on VLSI Systems*, 2016.
[17] F. Rittner *et al.*, "Automated test procedure to detect permanent faults inside SRAM-based FPGAs," in *NASA/ESA AHS*. IEEE, 2017.
[18] T. Slivinski *et al.*, "Study of fault-tolerant software technology," 1984.
[19] M. Liu and B. H. Meyer, "Bounding error detection latency in safety critical systems with enhanced execution fingerprinting," in *DFT*. IEEE, 2016.
[20] E. Wachter *et al.*, "A hierarchical and distributed fault tolerant proposal for noc-based mpsocs," *IEEE Transactions on Emerging Topics in Computing*, 2016.
[21] W. Liu, W. Zhang, X. Wang, and J. Xu, "Distributed sensor network-on-chip for performance optimization of soft-error-tolerant MPSoCs," *IEEE Transactions on VLSI Systems*, 2016.
[22] U. Martinez-Corral *et al.*, "A fully configurable and scalable neural coprocessor ip for soc implementations of machine learning applications," in *NASA/ESA AHS*. IEEE, 2017.
[23] S. S. Sahoo, B. Veeravalli, and A. Kumar, "Cross-layer fault-tolerant design of real-time systems," in *DFT*. IEEE, 2016.
[24] Y. Dong *et al.*, "COLO: Coarse-grained lock-stepping virtual machines for non-stop service," in *ACM Symposium on Cloud Computing*, 2013.
[25] G. Tsiligiannis *et al.*, "Testing a commercial MRAM under neutron and alpha radiation in dynamic mode," *IEEE Transactions on Nuclear Science*, 2013.
[26] C. M. Fuchs *et al.*, "Bringing fault-tolerant gigahertz-computing to space," in *IEEE ATS*, 2017.
[27] ——, "Towards affordable fault-tolerant nanosatellite computing with commodity hardware," in *IEEE ATS*, 2018.
[28] ——, "Dynamic fault tolerance through resource pooling," in *NASA/ESA AHS*. IEEE, 2018.
[29] M. Wirthlin, "High-reliability FPGA-based systems: space, high-energy physics, and beyond," *Proceedings of the IEEE*, vol. 103, no. 3, 2015.
[30] A. Samaras, F. Bezerra, E. Lorfevre, and R. Ecoffet, "Carmen-2: In flight observation of non destructive single event phenomena on memories," in *RADECS*.
[31] M. Darvishi *et al.*, "On the susceptibility of sram-based fpga routing network to delay changes induced by ionizing radiation," *IEEE Transactions on Nuclear Science*, 2019.
[32] P. Maillard *et al.*, "Single-event upsets characterization & evaluation of xilinx ultrascale™ soft error mitigation (sem ip) tool," in *REDW*. IEEE, 2016.
[33] A. P. Ferreira *et al.*, "Using pcm in next-generation embedded space applications," in *RTAS*. IEEE, 2010.
[34] D. S. Lee *et al.*, "Single-event characterization of 16 nm finfet xilinx ultrascale+ devices with heavy ion and neutron irradiation," in *NSREC*. IEEE, 2018.
[35] A. Geist *et al.*, "Spacecube v3.0 nasa next-generation high-performance processor for science applications," in *AIAA SmallSat*, 2019.
[36] B. Döbel, "Operating system support for redundant multithreading," Ph.D. dissertation, Dresden University, 2014.
[37] N. K. R. Beechu *et al.*, "Hardware implementation of fault tolerance NoC core mapping," *Springer Telecommunication Systems*, 2017.
[38] J. W. Lee, M. C. Ng, and K. Asanovic, "Globally-synchronized frames for guaranteed quality-of-service in on-chip networks," in *ACM SIGARCH*. IEEE, 2008.
[39] J. Zhou, H. Li, T. Wang, and X. Li, "Loft: A low-overhead fault-tolerant routing scheme for 3D NoCs," *Integration, the VLSI Journal*, 2016.