ScOSA on the Way to Orbit: Reconfigurable High-Performance Computing for Spacecraft

Daniel Lüdtke*, Thomas Firchau†, Carlos Gonzalez Cortes*, Andreas Lund‡, Ayush Mani Nepal*, Mahmoud M. Elbarrawy‡, Zain Haj Hammadeh*, Jan-Gerd Meß†, Patrick Kenny‡, Fiona Brömer*, Michael Mirzaagha‡, George Saleip*, Hannah Kirstein†, Christoph Kirchhefer†, Andreas Gerndt*§

*Institute for Software Technology
German Aerospace Center (DLR)
Braunschweig, Germany

†Institute of Space Systems
German Aerospace Center (DLR)
Bremen, Germany

‡Institute for Software Technology
German Aerospace Center (DLR)
Weßling, Germany

§University of Bremen
Bremen, Germany

***Abstract*—The German Aerospace Center (DLR) is developing ScOSA (Scalable On-board Computing for Space Avionics) as a distributed on-board computing architecture for future space missions. The ScOSA architecture consists of commercial off-the-shelf (COTS) and radiation-tolerant nodes interconnected by a SpaceWire network. The system software provides services to enable parallel computing and system reconfiguration. This allows ScOSA to adapt to node errors and failures that COTS hardware is susceptible to in the space environment. In the ongoing ScOSA Flight Experiment project, a ScOSA system consisting of eight Xilinx Zynq systems-on-chip with dual-core ARM-based processors and a LEON3 radiation-tolerant processor is being built for launch on DLR's next CubeSat in late 2024. In this flight experiment, not only all 18 cores but also the programmable logic will be used for high performance on-board data processing. This paper presents the current hardware and software architecture of ScOSA. The scalability of ScOSA is highlighted from both hardware and software perspectives. We present benchmark results of the ScOSA system and experiments of the ScOSA system software on ESA's OPS-SAT in orbit in combination with a machine learning application for image classification.**

***Index Terms*—COTS, FPGA, SpaceWire, Multi-core, Computer Architecture, Hybrid, LEON3, Reconfiguration, Benchmark**

I. Introduction

Spacecraft computers face unique challenges due to the harsh environment of space. This has led to the development of special versions of processors and other electronic circuitry that are tolerant of the radiation and thermal stresses of the environment. Typically, these specialized processors provide significantly less computing power than commercial off-the-shelf (COTS) components used in terrestrial embedded systems such as cars or smartphones. In addition, the cost of radiation-tolerant components is typically much higher than COTS components. On the other hand, the need for much more on-board processing power is also evident in the space domain. The amount of sensor and image data generated by new mission concepts and the still limited communication bandwidth require a considerable amount of on-board data processing (OBDP) to achieve the mission objectives. This is especially true with respect to the capabilities of deep learning networks that have been proposed for use in OBDP applications, e.g. [1]. Increased autonomy requirements for deep space robotic missions also demand high computational power, as demonstrated by Ingenuity's flights on Mars [2].

The Scalable On-board Computing for Space Avionics (ScOSA) architecture is a hybrid distributed computing architecture that provides high performance on-board processing while maintaining high reliability compared to traditional on-board computing approaches. The ScOSA architecture consists of a set COTS-based high-performance nodes (HPN) and radiation-tolerant reliable computing nodes (RCN) [3]. All nodes are connected by a SpaceWire network. The ScOSA system software provides services to enable parallel computing and system reconfiguration. This allows ScOSA to adapt to node errors and failures that COTS hardware is prone to, and to reconfigure for different mission phases that require different applications at different times.

An important design goal of the ScOSA architecture is scalability. The number of nodes and the node architecture are adaptable to mission requirements. ScOSA systems can be built with a minimum of three HPNs, two RCNs, one RCN combined with two HPNs, or with many more nodes of different types. The *ScOSA Flight Experiment*, scheduled for launch in late 2024 on a 12U CubeSat, will combine eight HPNs and one RCN. The ScOSA architecture is intended for spacecraft or rovers with high computational requirements that have sufficient power and thermal capacity to operate the required number of nodes. The hardware is designed to be easily scaled to the physical dimensions of CubeSats or larger satellite structures.

ScOSA has already come a long way on its path to orbit. The first activities started in 2012 and were continued until 2016 in the On-Board Computer-Next Generation (OBC-NG) project [4]. The project focused on the development of reconfigurable COTS-based high-performance on-board computers for future missions. The research continued with the ScOSA project [3], which proposed and developed an on-board computer architecture that combined radiation-tolerant hardware with the OBC-NG components and demonstrated a variety

Published Paper:
D. Lüdtke et al., "ScOSA on the Way to Orbit: Reconfigurable High-Performance Computing for Spacecraft," *2023 IEEE Space Computing Conference (SCC)*, Pasadena, CA, USA, 2023, pp. 34-44, doi: 10.1109/SCC57168.2023.00015.

of on-board applications from robotics, navigation, command and data handling, and image processing. A fault-tolerant and reconfigurable middleware was developed to ensure reliability and performance. Since 2020, the *ScOSA Flight Experiment* has been working to improve the technology readiness level (TRL) of ScOSA [5] in order to test it in orbit. As part of this effort, ScOSA has demonstrated the ability to run on-board data analysis applications in parallel [6].

During the in-orbit testing of ScOSA, benchmark applications will be run to evaluate the performance of the computer and the fault detection and mitigation techniques. In addition, the following four on-board applications will demonstrate ScOSA's capabilities:

1) An On-board Data Analysis and Real-time Information System (ODARIS) that includes an Iridium modem to directly alert users about relevant observations in processed camera images [7].
2) An on-orbit rendezvous navigation experiment that processes (simulated) LIDAR and camera images [8]. The experiment includes the ground control to test the full process of a rendezvous for an on-orbit servicing scenario.
3) An on-board image data compression experiment using the programmable logic of the FPGA in the HPNs [9].
4) A single event upset (SEU) detection and mitigation experiment on the HPNs to determine SEU rates and evaluate fault tolerance mechanisms.

In this paper, we present the hardware and software architecture of ScOSA with a focus on the ScOSA Flight Experiment and show first results of some preliminary tests.

The remainder of the paper is structured as follows: in Section II, we present some related work. In Section III, we describe the ScOSA system in terms of its hardware and in Section IV in terms of its software components, followed by the results of two different experiments we performed with the ScOSA system software in Section V. Finally, we conclude the paper and give an outlook in Section VI.

II. Related Work

The need for COTS-based on-board computers arose from two main requirements: the development of low-budget scientific satellites and autonomous missions that require high-performance computing that space-grade hardware cannot provide. COTS devices are typically more affordable than radiation-hardened hardware, but they are more vulnerable to the space environment. The effect of space radiation on COTS devices has been studied, for example, by Nikicio et al. [10]. [11] presented evaluations of COTS failure rates in space. Several approaches are being considered to enable COTS devices for the space environment. At the hardware level, shielding is possible, as well as extensive testing and careful selection of components [12]. Another approach, which is more independent of hardware components, is mitigation at software level.

The high performance capabilities of on-board computers are particularly important for applications that need to process image data, such as vision-based navigation described in [13] or the NGIS project described in [14]. To meet the high performance requirements, hybrid architectures combining radiation-tolerant processors with COTS components have been proposed. NASA has developed a hybrid architecture called High Performance, Dependable Multiprocessors [15]. The architecture combines one to two dependable processors and N independent high-performance COTS processors. The architecture is supported by a middleware that implements online load balancing strategies.

More recently, the multiMIND, CHICS, and Leopard Data Processing Unit (DPU) onboard computers were introduced [16]–[18]. These three solutions are high-performance, hybrid, fault-tolerant on-board computers based on radiation-hardened supervisors and the Xilinx Zynq Ultrascale+ COTS system-on-chip. They implement a number of fault-tolerance techniques such as SEU and single event latch-up detection and mitigation, software and hardware isolation, ECC-protected memory, and hardware redundancy, among others. On the other hand, the De-RISC and ICU4SAT solutions are based on RISC-V fault-tolerant processors and FPGA-based data processing units [19], [20]. The ICU4SAT project also introduces the use of FPGA-GPUs with the advantages of reconfigurability and a unified application development workflow. Also, Vasileios et al. explored in [21] fault-tolerance techniques for integrating the Zynq FPGA and the Myriad Vision Processing Unit (VPU) as COTS devices to accelerate the payload processing. There is a clear need for high-performance, reliable, and cost-effective hybrid on-board computing solutions. Therefore, we propose a scalable hardware architecture with as many reliable and high-performance nodes as needed, creating a distributed system ideal for computationally expensive or highly parallel workloads.

In addition, reconfiguration-based solutions are being considered for autonomous systems to implement different functionalities with limited computational resources. Bubenhagen et al. presented in [22] a Dynamically Reconfigurable Processing Module (DRPM) between space-qualified processors and space-qualified SRAM-based FPGAs for the DPU within the Solar Orbiter PHI instrument. Our work is interested in static reconfiguration between COTS-based processors where the phases of the mission, i.e., the functionalities to be executed, are known and planned off-line. In addition, our solution uses reconfiguration as a fault-tolerant mitigation for node failures.

III. ScOSA Flight Experiment Hardware

The hardware configuration for the in-orbit demonstration of ScOSA consists of an RCN and a total of eight HPNs distributed across four Multi-HPN Modules (MHMs).

A. Modular System

ScOSA as a system is targeted for those spacecraft that need the computing power but can also provide sufficient electrical power. In such spacecraft, the equipment for each subsystem can usually also be grouped into one or more dedicated boxes, which are then mounted on the spacecraft structure.

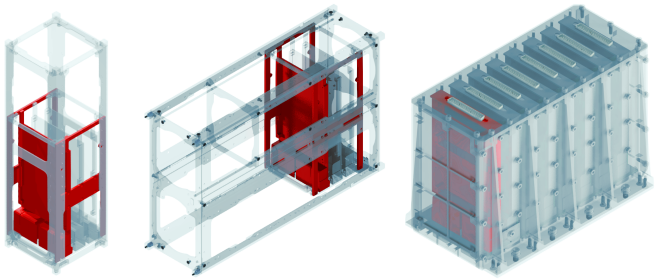


Fig. 1: ICA-UMF modules (red) in a 3U and 6U CubeSat structure as well as in a customer-specific housing.

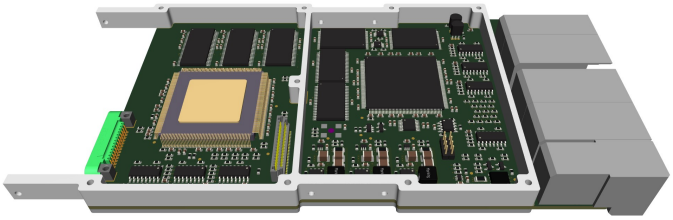


Fig. 2: Reliable computing node implemented in an ICA-UMF module for use in a CubeSat.

For the in-orbit demonstration of ScOSA, only the limited resources and accommodation space of a 12U CubeSat are available. In order to meet this challenge, the *Integrated Core Avionics - Unified Module Framework (ICA-UMF)* developed at the DLR will be used.

The Unified Module Framework is based on Compact PCI (CPCI) Serial Space and includes specifications for a set of module types, their mechanical characteristics, and their logical and electrical interconnections. The resulting avionics systems can be integrated into both CubeSats and stand-alone enclosures while maintaining compatibility with full-size CPCI Serial for Space systems [23]. Example integrations are shown in Fig. 1.

In comparison to CPCI Serial for Space, ICA-UMF defines a narrower module format with a reduction in width from 100 mm to 94 mm. The reduction in width comes at the expense of the thermal interface to the mechanics for conduction cooling. Given the small capacity of a CubeSat for cooling its subsystems, this is considered an acceptable trade-off. For the intended target applications in larger spacecraft with classic boxed equipment, the module width can be increased to improve thermal contact, or left as is and mounted on a different mechanical frame that adapts the module to the standard size. A visualization of an ICA-UMF module with an RCN is shown in Fig. 2.

CPCI Serial Space uses a backplane to connect modules. For use in a CubeSat, ICA-UMF features a smaller backplane but maintains the same interfaces between it and the modules. This ensures that modules can be tested with commercial CPCI Serial Space or non-space backplanes.

CPCI Serial Space also defines two logical module types. Up to two system modules and up to seven peripheral modules

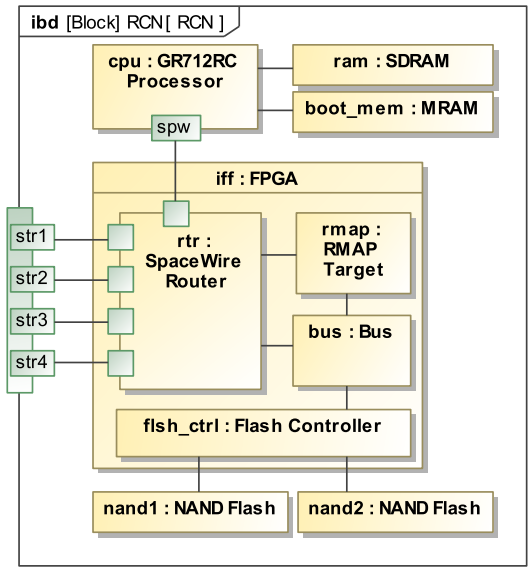


Fig. 3: RCN architecture.

can be used. The redundant system modules are each at the center of a separate star topology network. For the star network, CPCI Serial Space allows either PCI Express Serial, several Ethernet-based standards, or SpaceWire to be used. To avoid compatibility issues and simplify the module interfaces, ICA-UMF only uses SpaceWire as it is comparatively simple, robust, and widely used in the space domain.

In addition to the star network, an optional mesh network is available that provides a direct connection between each module. Again, CPCI Serial Space allows different interface technologies, but ICA-UMF limits the choice to SpaceWire for the time being.

For the ScOSA Flight Experiment, the four MHMs are peripheral modules and the RCN is the only system module.

B. Reliable Computing Node (RCN)

The reliable computing node is implemented by a radiation-tolerant LEON3 processor and a mix of military and automotive grade COTS parts. The COTS parts have been qualified for space through successful use in a payload on the Eu:CROPIS mission [24], [25]. Fig. 3 shows the important parts of the RCN architecture and Fig. 2 the physical implementation.

In addition to SDRAM, the processor is connected to small Magnetoresistive RAM (MRAM). The memory matrix of MRAM is radiation resistant and is used to store the critical boot loader.

A SpaceWire link between the processor and the FPGA provides access to the rest of the system. A Remote Memory Access Protocol (RMAP) target core in the FPGA provides access to the NAND flash controller. The connected NAND flash devices store multiple copies and versions of the flight application.

At power-up, the boot loader accesses the NAND flash via the SpaceWire link and the FPGA to locate and load the chosen version of the flight application. Once the flight

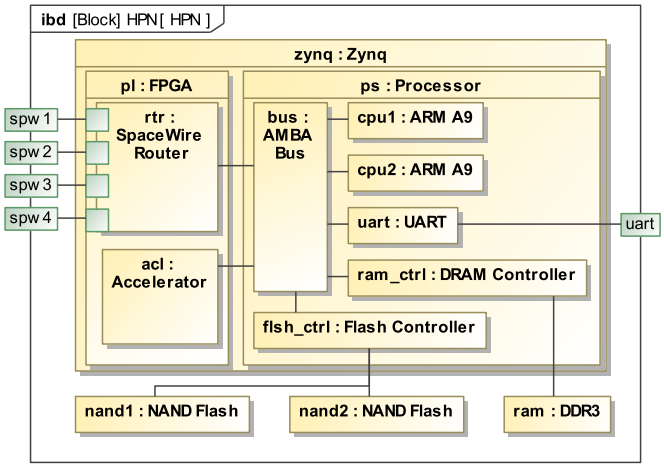


Fig. 4: HPN architecture.

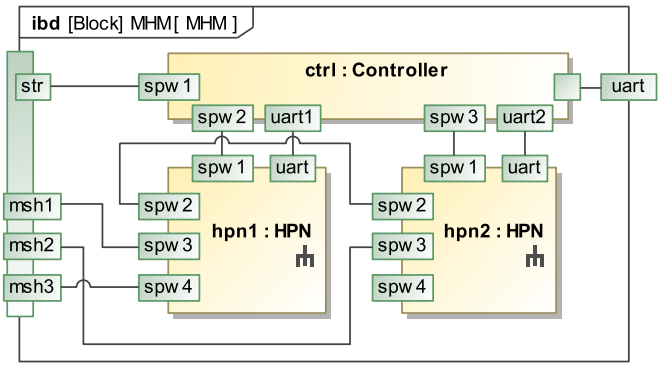


Fig. 5: MHM architecture.

application has been loaded into RAM and verified to be error free, it is handed over for execution. The flight application can then communicate with the rest of the system through the SpaceWire router.

C. High-Performance Nodes and Multi-HPN Module

The high-performance nodes each consist of a Xilinx Zynq 7020 System-on-Chip (SoC), which combines two ARM A9 general-purpose computing cores and an FPGA in one package, DDR3 RAM, and two NAND flash memories. The architecture of the HPN is shown in Fig. 4.

The FPGA part of the Zynq SoC is used to add a SpaceWire router and other external interfaces to the system. An internal SpaceWire interface to the router is accessible to the processor cores via the internal data bus. The FPGA also provides room for additional custom accelerator cores.

The HPNs provide significantly more processing power than the RCNs, but are considered less reliable. They are therefore isolated, grouped, and managed by an additional controller on Multi-HPN modules. Fig. 5 shows the architecture of an MHM with two HPNs.

The controller on the MHM can turn the HPNs on and off, monitor their currents and voltages, and provides an additional

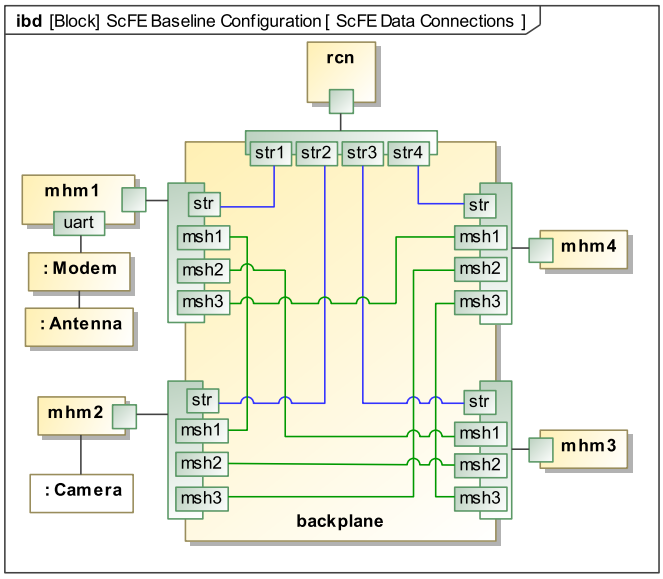


Fig. 6: SpaceWire network connections through the backplane.

SpaceWire router that connects the SpaceWire star network to the HPNs. The controller also provides a serial interface (UART) switch so that external peripherals can communicate through both HPNs.

The HPNs are also connected to each other and to the optional mesh network via the MHM's backplane connection.

D. SpaceWire Network

Both the Multi HPN modules and the RCN module provide multiple SpaceWire interfaces to the backplane. Fig. 6 shows how the backplane connects all the modules.

The blue lines represent the SpaceWire network with the star topology and the RCN at the center. The green lines represent the mesh network. Considering that within the modules there are also SpaceWire routers at each interface, there are multiple ways to reach each node on the network, even when individual interfaces or nodes fail.

Fig. 6 also shows external devices connected to the system. For the ScOSA Flight Experiment, a real-time communication modem with an externally mounted antenna and a camera are planned.

IV. ScOSA System Software

The ScOSA system software enables and unleashes the full potential of the distributed architecture described in the previous section. In the following, we present the overall architecture of the entire software stack as well as the architecture of the middleware. We also explain the basic fault-tolerance and resilience mechanisms and briefly describe how we generate different configurations and configuration trees using a model-based systems engineering tool.

A. Middleware & Software Stack

The ScOSA middleware consists of three main components [5]:

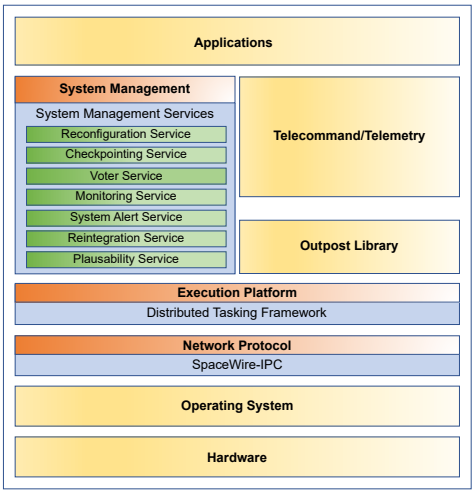


Fig. 7: ScOSA's layered software stack [5].

- *SpaceWireIPC* is a reliable message-passing protocol designed to operate over SpaceWire or Ethernet links [26]. It provides Inter-Process Communication (IPC) over SpaceWire. The protocol supports error detection and handling services, including timeouts and delivery acknowledgments, and packet fragmentation for seamless transmission of large messages from the application layer.
- *Distributed Tasking Framework* is a distributed version of the Tasking Framework [27] that allows tasks to be seamlessly executed on remote nodes. This allows computationally intensive tasks to be delegated to more powerful computing nodes or distributed across multiple nodes. Communication is realized by SpaceWireIPC.
- *System Management Services* are a set of services designed to provide Fault Detection, Isolation and Recovery (FDIR) techniques to the middleware. They include the Reconfiguration Service, Monitoring Service, Voting Service, Updates Service, and Telecommand/Telemetry Service. The services are mainly implemented as threads and use the SpaceWireIPC protocol to communicate with the rest of the nodes and keep the distributed system state consistent.

SpaceWireIPC and the Distributed Tasking Framework are being developed as part of ScOSA. Fig. 7 shows a layered view of the ScOSA software stack. User applications are built on top of the execution platform and system services. Application designers use the tasking model to build distributed applications using the communication and activation capabilities of the Tasking Framework. The Reconfiguration Service stores information about task-node mappings, i.e. configurations, for different system states. Configurations are stored as a tree and are automatically applied in the event of node failures.

The execution platform is based on two proven technologies: Tasking Framework and OUTPOST. The Tasking Framework provides an event-driven, multi-threaded task execution mechanism [27]. On the other hand, OUTPOST [28] is a modular library that provides a set of common low-level space system functionalities and the abstraction layer to different operating systems and hardware platforms. Therefore, ScOSA supports GNU/Linux and the RTEMS real-time operating systems. OUTPOST also provides a set of standardized communication protocols for integration into a variety of satellite buses or even as a standalone product with self-managed space-to-ground communication. It is currently successfully used in several DLR satellites, space and launcher missions, including Eu:CROPIS and MMX [29].

B. Fault-Tolerance & Resilience

ScOSA aims to increase the reliability of the spacecraft by incorporating several fault-tolerance and resilience techniques.

The most important of these techniques is ScOSA's ability to reconfigure its task distribution among its distributed nodes. This *reconfiguration* ensures that a node failure can be compensated at any time by reassigning its tasks to the remaining nodes. However, it is not limited to compensating for node failures, but is also capable of reassigning a complete new set of tasks to the distributed system. This feature allows ScOSA to support different mission phases with the same hardware. The reconfiguration mechanism is implemented by the Reconfiguration Service in cooperation with the Monitoring Service and the SpaceWireIPC module.

SpaceWireIPC provides the feature of reliable messaging, which means that a transmission must be acknowledged by the receiver. If the transmission is not acknowledged, SpaceWireIPC retransmits it three times. After the third timeout for a response, an alarm is triggered indicating a node-loss failure. This mechanism is used by the monitoring service to observe if the nodes in the system are still responding. For this reason, the service sends heartbeat requests as reliable messages that must be acknowledged by the receivers. We leave the ability to monitor the nodes to a special node, called the coordinator. To monitor the health of the coordinator, an observer role has been introduced that repeats the above procedure by sending a heartbeat request to the coordinator. All other nodes are internally addressed as worker nodes.

The sequence of a reconfiguration due to a node failure is as follows (see Fig. 8): When a node fails, the Reconfiguration Service is notified with an error notification sent by the SpaceWireIPC protocol handler. The service will then inform all other nodes of the failed node with a *UpdateFailedNode* message, immediately followed by a *RequestReconfiguration* message containing the ID of the task configuration to which the nodes should switch. Such a configuration contains the mapping of tasks to nodes. Many of these configurations must be kept in the middleware to cover as many scenarios as possible where nodes may fail. After informing all nodes of the new configuration, the issuing coordinator waits for *FinishReconfiguration* messages from all still-running nodes indicating that the nodes have switched to the requested configuration. Meanwhile, the coordinator itself changes the configuration by finishing all tasks that are still running, then disabling all connections between tasks and channels so that no task is triggered for execution, and finally establishing the

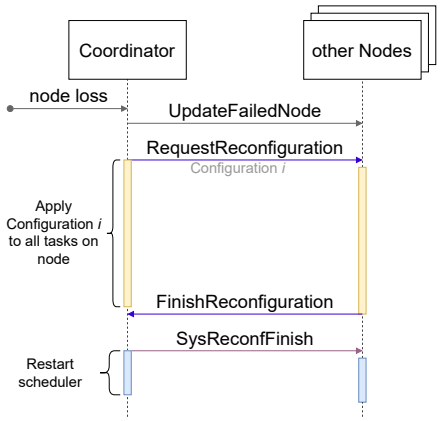


Fig. 8: The messaging sequence between the coordinator and the other nodes when a node fails.

connections between the tasks and channels that are assigned to it according to the new configuration. Once all nodes have applied the new configuration, the coordinator will send *SysReconfFinish* messages to all nodes, which will restart their task schedulers upon receipt.

The entire reconfiguration process is based on precomputed configurations. This approach was chosen to ensure determinism and traceability as well as speed compared to online strategies. All nodes have the possible configurations stored in a tree structure, where the tree nodes indicate the configurations and the edges indicate failed nodes. The transition from a node $C1$ along an edge $N1$ to a node $C2$ represents the transition from configuration $C1$ to $C2$ when node $N1$ fails. When a node fails, the selection of the next configuration during runtime is very fast. In addition, it is easy for ground operators to track what changes have been applied, as the pre-computed configuration tree is also available on the ground. The disadvantage of this approach is that all scenarios to be covered must be precomputed and included in the ScOSA system software, which consumes memory. In addition, if an unforeseen situation occurs, the reconfiguration algorithm cannot handle it. It will switch to a safe mode and wait for ground interaction. However, resilience requires that the system also responds to unanticipated failures and their effects by maintaining its availability. This is not provided by a safe mode approach, so we investigate the possibility of an online reconfiguration algorithm. However, such an online algorithm may be slower than its offline counterpart, and the configuration changes will be more difficult for ground operators to track.

Triple Modular Redundancy (TMR) is a well-known and widely used technique to counteract soft errors in tasks. A task is executed three times, either in parallel on different hardware or sequentially on the same hardware. The three results are then compared and the majority value is selected. ScOSA offers the same principle with the Voter Service, but multiple instances, not just three, of the same task can be run and their results compared. The service will send an alarm if no majority can be found among the instances.

If a node fails while executing a task, the current state of that task is lost. After reconfiguration, the task would start on a new node with a completely new state. To prevent this loss of information, ScOSA provides the Checkpoint Service. It periodically receives the state of a task, stores it in a non-volatile memory of the hardware and additionally sends it to another node, which also stores the state in its non-volatile memory. The receiving node is selected using the configuration tree mentioned above. The service checks which configuration will be selected if its own node fails and selects the node from the new configuration to which the corresponding task will migrate after this reconfiguration.

C. Model-based Systems Engineering

As described above, ScOSA's reconfiguration mechanism is an offline and static approach. All possible configurations (task-to-node mappings) for the different node states are stored in memory within the middleware. When a node state changes, the appropriate configuration is selected by the coordinator node, which then informs all other nodes of the configuration change. Creating all these configurations covering the many different combinations of node states is cumbersome, especially when ScOSA has to support many mission phases with different task to node mappings. For this reason, we have developed an automatic configuration generator. The generator solves a combinatorial optimization problem of mapping tasks to processing nodes for every node either with an SMT solver or by using a genetic algorithm while avoiding node overload [30]. The tool generates a dedicated configuration tree for each mission phase. The automatic configuration generator is developed as an extension of DLR's model-based systems engineering tool Virtual Satellite (see Fig. 9).

V. First Results

The first flight of a ScOSA on-board computer is planned for the end of 2024. In addition to the space applications mentioned in Section I, a number of benchmarks will be performed to characterize the system and test all FDIR mechanisms. In this section, we present two activities to prepare for the ScOSA Flight Experiment in 2024 and to pre-qualify parts of the software framework. First, a set of benchmarks is presented and second, an in-orbit experiment of the ScOSA software framework on ESA's OPS-SAT is described.

A. Benchmarks

The ScOSA Flight Experiment system consists of eight Xilinx Zynq 7020 SoC with dual-core ARM Cortex A9 (ARMv7-A) processors at 886 MHz and 1.0 GB RAM. For preliminary results, a ScOSA software development model with three Zynq 7020 modules is used.

We use a set of benchmarks to evaluate the available computing power and to establish a baseline for future comparison. Space applications can vary in nature and complexity, so we selected a set of general purpose and space-domain benchmarks and evaluated their applicability to ScOSA [31].

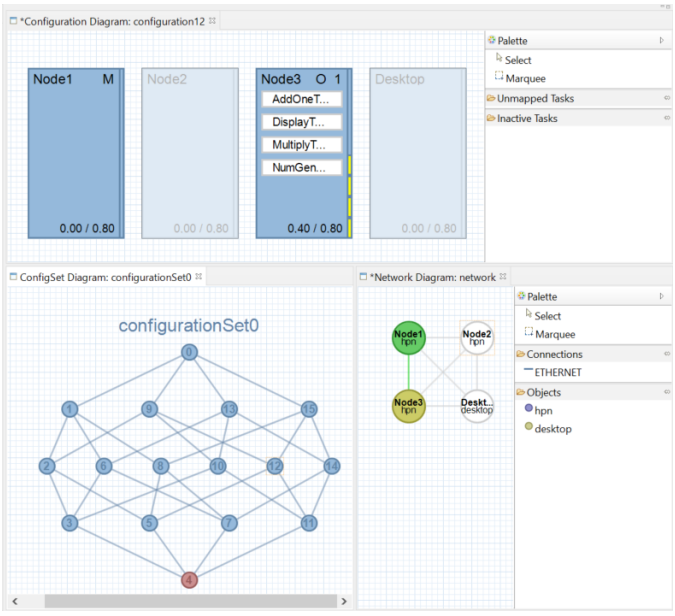


Fig. 9: The automatic configuration generator as an extension of the model-based systems engineering tool Virtual Satellite.

TABLE I: Benchmark (BM) comparison.

BM	Relevance	Verifiability	Scientific popularity	Openness	Access to reported data	Type
Whetstone	0	+ +	+ +	+ +	+ +	Synthetic
Dhrystone	0	+ +	+ +	+ +	+ +	BMs
LINPACK	0	+ +	+ +	+ +	+ +	Application
LAPACK	0	+ +	+ +	+ +	+ +	BMs
TPC	- -	+	+	- -	0	General
SPCE	- -	+	+	- -	0	purpose
GeekBench	- -	+	+	+	+ +	CPU BMs
MiBench	0	+	+	+ +	0	
MediaBench	0	+	+	+ +	0	Embedded
BDTI	0	0	0	- -	-	BMs
EEMBC	0	+	+ +	0	+	
GPU4S	+ +	+	+	+ +	+ +	Space
OBPMark	+ +	+	+	+ +	+ +	application
						BMs

(+ +): Excellent. (+): Good. (0): Neutral. (-): Bad. (- -): Worse

Therefore, according to Table I, we selected the following benchmarks: LINPACK, Dhrystone, and OBPMark [32].

For each benchmark, our evaluation methodology is as follows:

- Design and implement a single-node ScOSA application
- Design and implement a distributed ScOSA application
- Run the single node and distributed applications locally on our reference machine[1]
- Run the single node and distributed applications on the HPN's software development model board[2]
- Analysis results

In the following, a brief summary of the first results of the study is presented.

1) LINPACK: This benchmark assesses the performance by generating a dense system of linear equations and measuring how much time it takes to solve this matrix. The measuring unit is floating point operations per second (FLOPS) [33].

We are interested in the accumulated performance of the system. Therefore, we need to parallelize LINPACK. Our parallelization strategy is to create a LINPACK task to execute the main loop of the benchmark and assign this task to each node. Since the LINPACK benchmark score is based on the rate of random floating-point operations, we consider the operations of the nodes to be independent; thus, we benchmark each CPU individually. A result collector task calculates how many operations per second were achieved by adding up each CPU result as shown in Fig. 10.

The execution results are shown in Table II. They give us reference values for the processing power available on the tested three-node ScOSA computer. They are 320 MFLOPS for single and 200 MFLOPS for double precision, respectively.

2) Dhrystone: Dhrystone is a non-numeric synthetic benchmark that measures source language features of system-type programming. System-type programming includes fewer loops, simpler computations, and more branches and function calls. Thus, Dhrystone evaluates the integer performance of processors. The unit of measurement is Dhrystone millions of instructions per second (DMIPS) [34].

In this case, our parallelization strategy is to implement a map-reduce scheme in ScOSA. Thus, we distribute the Dhrystone independent workload across multiple nodes. As shown in Fig. 11, the SplitTask distributes and synchronizes the remote nodes, then each node executes the corresponding number of Dhrystone loops, and finally the ReduceTask receives the partial results and computes the global performance measures.

The benchmark results are shown in Table III. We see that the Zynq 7020 embedded system achieves 1983 DMIPS. ScOSA's ability to distribute the workload results in a speedup

[1]18 core Intel(R) Core(TM) i9-10980XE CPU @ 3.00GHz, 128GB RAM, Ubuntu 20.04.5 LTS

[2]3 x Zynq 7020 SoC, dual core ARM Cortex A9 CPU @ 886 MHz, 1 GB RAM, Xilinx PetaLinux v2020.2

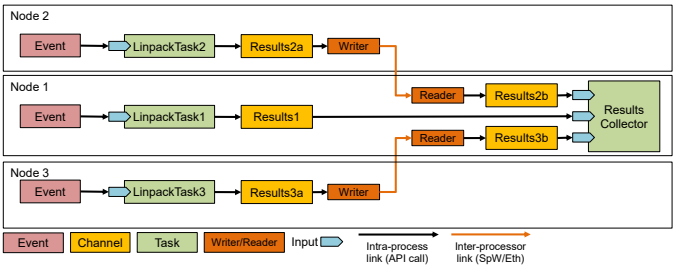


Fig. 10: Distributed version of the LINPACK benchmark.

TABLE II: LINPACK benchmark results.

Machine	Nodes	LINPACK Single prec. MFLOPS	LINPACK Single prec. Ratio	LINPACK Double prec. Mflops	LINPACK Double prec. Ratio
Reference	1	4553	1.00	3515	1.00
	2	8799	1.93	6923	1.97
	3	13063	2.87	10035	2.85
HPN	1	107	1.00	67	1.00
	2	215	2.01	134	2.00
	3	321	2.99	201	3.01

Problem size N=1000. Values shown are the average of three runs.

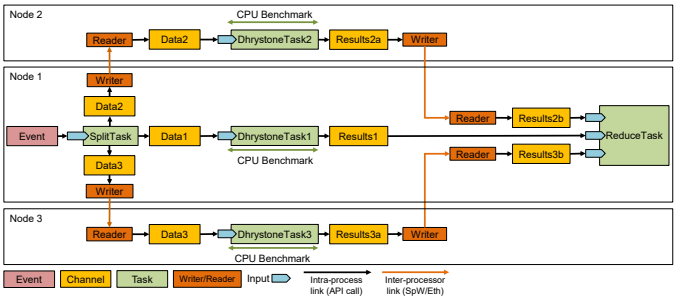


Fig. 11: Distributed version of the Dhrystone benchmark.

TABLE III: Dhrystone benchmark results.

Machine	Nodes	Dhrystone/second	DMIPS	Ratio
Reference	1	58,548,383	33323	1.00
Reference	3	175,191,437	99711	2.99
HPN	1	3,483,188	1983	1.00
HPN	3	10,400,857	5920	2.99

Dhrystone loops = 500,000,000. Values shown are the average of three runs.

of 2.99 when the benchmark is run on three nodes.

3) OBPMark: The European Space Agency and the Barcelona Supercomputing Center work on a set of benchmarks called OBPMark (On-Board Processing Benchmarks) that encompass software often used on spacecraft [35]. In particular, we are using the Benchmark #1.1: Image Calibration and Correction [36].

The original image processing workload consisted of the pipeline shown in Fig. 12. The input is a set of frames (8 images) and the output is only one processed frame. The measured result is the total processing time. The parallelization strategy also includes a map-reduce operation, where the same pipeline in Fig. 12 is applied to different sections of the input images. As shown in Fig. 13, the input images are split horizontally into as many parts as available processing nodes. The split has to consider an overlapping area because some

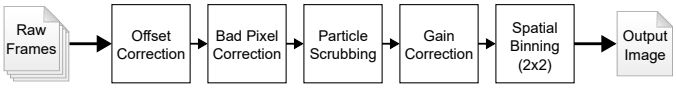


Fig. 12: OBPMark image processing workload (image adapted from [36]).

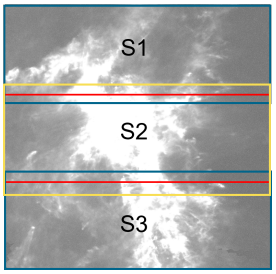


Fig. 13: Image split strategy for the distributed OBPMark benchmark [32]. S1, S2 and S3 represent the subimages when splitting for three nodes.

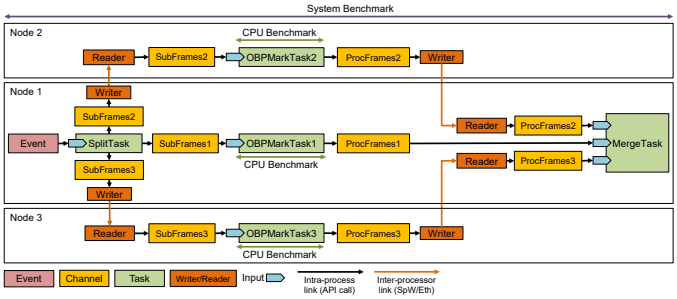


Fig. 14: Distributed version of the OBPMark benchmark.

TABLE IV: OBPMark Image Processing benchmark results.

Machine	**Nodes**	**OBPMark Task execution (milliseconds)**	**OBPMark MPixels/s**	**OBPMark Ratio**
Reference	1	374	22.43	1.00
	3	449	18.63	0.83
HPN	1	8660	0.97	1.00
	3	4390	1.91	1.97

Processed pixels = 8x1024x1024 = 8.389 MPixels.
Values shown are the average of three runs.

of the pipeline steps require the pixel surrounding values to calculate the output. This is considered in the design of the distributed application shown in Fig. 14.

The execution results are reported in Table IV. We can see an improvement in the processing time for the HPN modules, but not on the reference machine. To better understand these results, we present detailed time measurements in Table V.

It shows the time required to execute each task and send image data through channels. We can observe that it takes about 230 ms to transmit the approx. 12 MiB of input data to each task on the remote nodes on the reference machine and about 1450 ms on the HPN. Thus, the measured data rates are 52.17 MiB/s and 8.27 MiB/s respectively. Therefore, the overhead of sending input data to the remote nodes and sending the partial results back to the first node is significant. Moreover, on the reference machine, the task execution time is considerably lower than the time required to send data; thus, the overall execution time is heavily affected by this overhead, and we see a decreasing performance when the workload is

TABLE V: OBPMark benchmark detailed execution times.

	Reference milliseconds	**HPN milliseconds**
SubFrames1 channel Tx time*	3	33
SubFrames2 channel Tx time*	230	1295
SubFrames3 channel Tx time*	233	1450
ProcFrames1 channel Tx time†	259	1219
ProcFrames2 channel Tx time†	18	195
ProcFrames3 channel Tx time†	15	17
SplitTask execution time	54	293
OBPMarkTask1 execution time‡	113	2743
OBPMarkTask2 execution time‡	121	2382
OBPMarkTask3 execution time‡	195	2355
MergeTask execution time	16	20
Total Execution time	**449**	**4390**

Values shown are the average of three runs. Total execution time is the critical path along node 1.
*Input data size = 12185644 bytes
†Output data size = 716832 bytes
‡Number of processed pixels (frames x height x wdth) = 8 x 1024 x 1024

distributed. The HPN case is similar, but the task execution time and the data transmission time are in the same order of magnitude, so we can still see a performance improvement.

4) Discussion: The preliminary set of benchmarks selected and adapted to run as ScOSA applications provide relevant information about the available computing power. LINPACK and Dhrystone are commonly used to characterize and compare computer systems, although they are not representative space applications compared to OBPMark. Thus, the results presented provide now a baseline against which to compare further improvements in hardware and software, and to understand how the space environment affects performance. These results also demonstrate ScOSA's ability to distribute workloads across remote nodes for parallel processing. Our primary interest was to benchmark the Zynq 7020 CPU. For this purpose, we considered the LINPACK and Dhrystone benchmarks as an independent workloads: Thus, we reported the overall system performance as the sum of each CPU metric. Further testing is required to include the distributed system synchronization overhead in the performance and to report more realistic scalability results.

However, the OBPMark Image Processing Benchmark is a good example of a more realistic space application workload. We have designed a generic map-reduce ScOSA application and developed a distributed version of the OBPMark Benchmark #1.1: Image Calibration and Correction. A key difference from the other selected benchmarks is that the image processing benchmark requires realistic data distribution to remote nodes. This data transfer adds significant overhead to the overall execution time and exposes a bottleneck in our network stack implementation. Thus, the performance metrics presented are the baseline for understanding the effect of further improvements in the middleware.

In further activities, we will develop benchmarks to assess and evaluate the fault tolerance and resilience of ScOSA. For this purpose, we will collect metrics that can be used to benchmark a fault tolerance mechanism, such as the time and memory consumed by the technique, commonly known as overhead. These benchmarks will then be used to compare the aforementioned online and offline reconfiguration algorithms.

B. In-orbit Software Experiments on ESA's OPS-SAT

ESA's OPS-SAT mission [37] provides a unique opportunity to test new software ideas and operational concepts directly in orbit. OPS-SAT is designed to establish a reliable and robust data processing platform on a satellite using COTS hardware and open source software. The OPS-SAT computing platform consists of an Altera Cyclone V SoC with an ARM dual-core Cortex-A9 MPCore processor running Linux. It also includes a Cyclone V FPGA. This configuration in OPS-SAT facilitates the safe execution of open source software and allows the implementation of high-level programming languages such as Java and Python to control and monitor the entire satellite.

The OPS-SAT platform was used as a testbed to evaluate the reconfiguration logic and memory management capabilities of the ScOSA software. Given the constraint of having only a single processor available for experimentation on OPS-SAT, it was not feasible to fully test the reconfiguration logic using physically separate computing nodes. Nevertheless, the reconfigurability of the ScOSA software was evaluated using two virtual nodes implemented as separate processes on the same processor. To support this, a Java application was developed that implemented the NanoSat Mission Operations Framework (NMF) provided by ESA for the purpose of monitoring and controlling the satellite. This Java application was responsible for setting the desired attitude mode, initiating the camera, and forwarding captured images to the ScOSA middleware using the SpaceWireIPC protocol.

In addition to evaluating the reconfiguration logic, the memory management capabilities of the ScOSA software were tested by implementing neural network inference on satellite images. This was done to enable selective downlinking of images. The varying lighting conditions in space can result in satellite images that are not visually informative. Therefore, in order to optimize the limited downlink bandwidth, it is necessary to pre-select data before sending it.

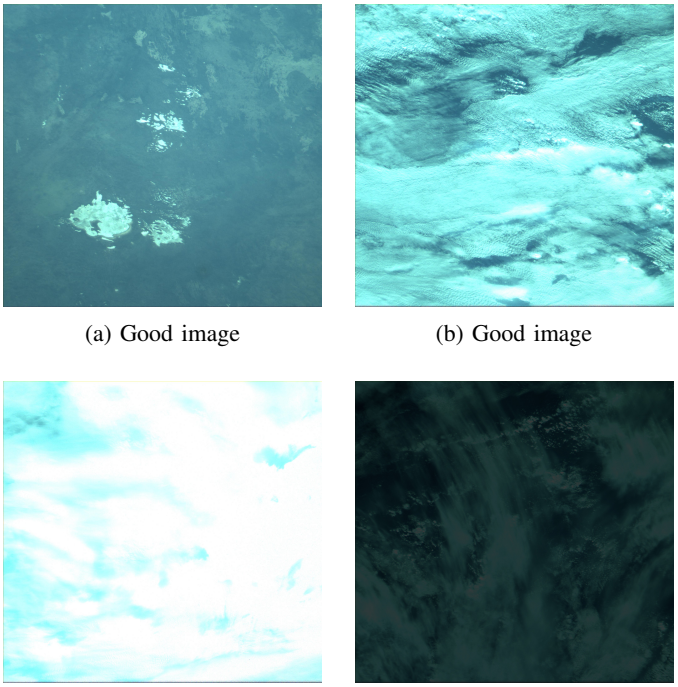
1) Experiment 1: In the first experiment, a single instance of the ScOSA software was executed, consisting of three tasks: the Receiver task, the Classifier task, and the Logger task. The Receiver task received the camera images transmitted by the Java application and passed them to the Classifier task. The Classifier task processed the received images by passing them through a binary Convolutional Neural Network (CNN) classifier, which categorized the images as *good* or *bad*. The logger task recorded the classification results.

The CNN classifier model was trained on the ground before being transmitted to the satellite. Since training such image classifiers typically requires a large dataset, we used the transfer learning technique. Transfer learning involves reusing and building on a generic model that has been previously trained on large datasets. We used the MobileNet V2 CNN as the feature extractor, which was pre-trained on the ImageNet image dataset [38]. We fine-tuned the model using an archive of about 4000 images previously downlinked from OPS-SAT. Finally, to reduce the size and inference latency of the CNN model, it was quantized to 16-bit floating-point precision.

Out of nine images taken by OPS-SAT during the on-orbit experiment, five were correctly classified as *bad* and three were correctly classified as *good*. One *bad* image was misclassified as *good*. Examples of both types are shown in Fig. 15.

2) Experiment 2: The second experiment focused on evaluating the reconfigurability of the ScOSA software. For this purpose, two instances of ScOSA were executed as separate processes, simulating the execution on two virtual computing nodes. The experiment was divided into two mission phases. A mission phase corresponds to a set of tasks that are performed when ScOSA has been configured for that phase.

The first mission phase consisted of the image acquisition tasks. After approximately 35 minutes of execution, the ScOSA system transitioned to the second mission phase in which 20 instances of a Ping task and 20 instances of a Pong



(a) Good image (b) Good image

(c) Bad image (high exposure) (d) Bad image (too dark)

Fig. 15: Images taken by OPS-SAT during the ScOSA experiments.

task were initiated. Each task sent and received one byte of data to and from its counterpart. Initially, all Ping tasks ran in one process, and all Pong tasks were configured to run in the other process, simulating a two-node scenario.

After one minute of execution, one of the ScOSA processes was deliberately terminated, simulating a failure of the virtual compute node. In response, ScOSA automatically reconfigured itself to continue execution without interruption, transferring all 20 Pong tasks to the still-running ScOSA instance. This demonstrated ScOSA's ability to adapt to a node failure and maintain service continuity.

3) Discussion: The ScOSA experiments on OPS-SAT in orbit, and the much larger ground-based experiments in preparation, have achieved the following goals:

1) The ScOSA system software was successfully operated in a relevant environment.
2) The neural network image processing application demonstrated that the class of processors is capable of running this type of application.
3) The ScOSA middleware was shown to be able to successfully process large data packages such as images.
4) The reconfiguration feature was successfully demonstrated with planned reconfiguration and fault recovery.

All of the experiments were also performed on the ground, but putting these preliminary experiments of the actual ScOSA Flight Experiment into orbit was a very good exercise to define concepts of operations, potential challenges of controlling a real CubeSat during the experiments, the interface to the operations team, planning and performing software updates, and much more.

VI. Conclusions and Outlook

This paper presented the current state of the ScOSA on-board computer architecture and its path to in-orbit demonstration. The heterogeneous and distributed nature of ScOSA allows the combination of reliable, radiation-tolerant processors with high-performance, COTS-based multicore SoCs in combination with FPGA-based co-processors. This hardware architecture, together with the ScOSA system software that enables rapid reconfiguration of active compute nodes and task-to-node mapping, enables more autonomous on-board computing applications by achieving high reliability.

The architecture is highly scalable from two reliable nodes to a cluster of COTS-based multi-core SoCs. This means that ScOSA can be tailored to specific mission requirements. It also allows rapid adaptation to new hardware platforms (e.g. RISC-V) and to the needs of applications in terms of real-time requirements and development libraries. We envision ScOSA for spacecraft or rovers of about 12U or larger.

The experiments conducted showed that the ScOSA architecture provides the computational resources to run complex applications on a spacecraft, and the FDIR mechanisms including the reconfiguration scheme ensure high reliability of the system.

Due to the scalability of ScOSA, we have the opportunity to test a minimal system consisting of our basic hardware components (one RCN and one MHM, including two HPN) with the SEU detection application on a 6U CubeSat mission in the first half of 2024. The full in-orbit demonstration on a 12U CubeSat targeted for launch in late 2024 will test a larger system as well as a number of different on-board applications with a strong focus on on-board data processing.

Besides the selected applications, it is planned to provide an API for additional applications that can be uploaded during the mission. As an outreach activity, we gave students the opportunity to develop small applications for the ScOSA flight experiment that could be executed in orbit. Furthermore, we plan to investigate the effects of using hypervisors in the nodes. This would allow to keep mixed criticality guarantees even in case of a reconfiguration where a critical and a non-critical task will be migrated to the same node.

Acknowledgment

The authors would like to thank all current and former project members of the ScOSA Flight Experiment and its predecessor activities. We would also like to thank all our colleagues in the institutes and the administration who supported the idea of a new on-board computer architecture. Without their ideas, contributions, funding and support, ScOSA Flight Experiment would not be possible. The authors also want to thank the OBPMark Team for providing early access and supporting the project. We are also grateful for the opportunity provided by ESA to test parts of ScOSA on OPS-SAT.

References

[1] G. Giuffrida, L. Diana, F. de Gioia, G. Benelli, G. Meoni, M. Donati, and L. Fanucci, "Cloudscout: A deep neural network for on-board cloud

detection on hyperspectral images," *Remote Sensing*, vol. 12, no. 14, 2020.

[2] T. Tzanetos, M. Aung, J. Balaram, H. F. Grip, J. T. Karras, T. K. Canham, G. Kubiak, J. Anderson, G. Merewether, M. Starch, M. Pauken, S. Cappucci, M. Chase, M. Golombek, O. Toupet, M. C. Smart, S. Dawson, E. B. Ramirez, J. Lam, R. Stern, N. Chahat, J. Ravich, R. Hogg, B. Pipenberg, M. Keennon, and K. H. Williford, "Ingenuity Mars helicopter: From technology demonstration to extraterrestrial scout," in *2022 IEEE Aerospace Conference (AERO)*, March 2022.

[3] C. J. Treudler, H. Benninghoff, K. Borchers, B. Brunner, J. Cremer, M. Dumke, T. Gärtner, K. J. Höflinger, D. Lüdtke, T. Peng, E.-A. Risse, K. Schwenk, M. Stelzer, M. Ulmer, S. Vellas, and K. Westerdorff, "ScOSA - scalable on-board computing for space avionics," in *International Astronautical Congress (IAC), Bremen, Germany, Oct. 1-5*, 2018.

[4] D. Lüdtke, K. Westerdorff, K. Stohlmann, A. Börner, O. Maibaum, T. Peng, B. Weps, G. Fey, and A. Gerndt, "OBC-NG: Towards a reconfigurable on-board computing architecture for spacecraft," in *IEEE Aerospace Conference, Big Sky, MT, USA, March 1-8, 2014*. IEEE, 2014.

[5] A. Lund, Z. A. H. Hammadeh, P. Kenny, V. Bensal, A. Kovalov, H. Watolla, A. Gerndt, and D. Lüdtke, "ScOSA system software: The reliable and scalable middleware for a heterogeneous and distributed on-board computer architecture," *CEAS Space Journal*, Mai 2021.

[6] P. Kenny, K. Schwenk, D. Herschmann, A. Lund, V. Bansal, Z. A. Haj Hammadeh, A. Gerndt, and D. Lüdtke, "Parallelizing on-board data analysis applications for a distributed processing architecture," in *2nd European Workshop on On-Board Data Processing (OBDP2021), June 14-17, 2021*, Juni 2021.

[7] K. Schwenk and D. Herschmann, "On-board data analysis and realtime information system - status & outlook," in *Deutscher Luft- und Raumfahrtkongress 2022 (DLRK 2022)*, September 2022.

[8] H. Frei, M. Burri, F. Rems, and E.-A. Risse, "A robust navigation filter fusing delayed measurements from multiple sensors and its application to spacecraft rendezvous," *Advances in Space Research*, Oktober 2022.

[9] T. Freitag, "Acceleration of an autoencoder using a FPGA-SoC in a high-performance node of a distributed onboard computer," Master's thesis, TU Darmstadt, Dezember 2022.

[10] A. N. Nikicio, W.-T. Loke, H. Kamdar, and C.-H. Goh, "Radiation analysis and mitigation framework for leo small satellites," in *2017 IEEE International Conference on Communication, Networks and Satellite (Comnetsat)*, 2017, pp. 59–66.

[11] C. Wilson and A. George, "CSP hybrid space computing," *Journal of Aerospace Information Systems*, vol. 15, no. 4, pp. 215–227, 2018.

[12] D. Sinclair and J. Dyer, "Radiation effects and cots parts in smallsats," in *Small Satellite Conference*, 2013.

[13] G. Lentaris, K. Maragos, I. Stratakos, L. Papadopoulos, O. Papanikolaou, D. Soudris, M. Lourakis, X. Zabulis, D. Gonzalez-Arjona, and G. Furano, "High-performance embedded computing in space: Evaluation of platforms for vision-based navigation," *Journal of Aerospace Information Systems*, vol. 15, no. 4, pp. 178–192, 2018.

[14] D. Keymeulen, S. Shin, J. Riddley, M. Klimesh, A. Kiely, E. Liggett, P. Sullivan, M. Bernas, H. Ghossemi, G. Flesch, M. Cheng, S. Dolinar, D. Dolman, K. Roth, C. Holyoake, K. Crocker, and A. Smith, "High performance space computing with system-on-chip instrument avionics for space-based next generation imaging spectrometers (NGIS)," in *NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, 2018, pp. 33–36.

[15] J. Samson, J.R., E. Grobelny, S. Driesse-Bunn, M. Clark, and S. Van Portfliet, "Post-TRL6 dependable multiprocessor technology developments," in *Aerospace Conference, IEEE*, 2010.

[16] A. Pawlitzki and F. Steinmetz, "multiMIND–high performance processing system for robust newspace payloads," in *2nd European Workshop on On-Board Data Processing (OBDP2021)*, 2021.

[17] R. Costa Amorim, R. Martins, P. Harikrishnan, M. Ghiglione, and T. Helfers, "Dependable MPSoC framework for mixed criticality applications," in *2nd European Workshop on On-Board Data Processing (OBDP2021)*, 2021.

[18] P. Kuligowski, G. Gajoch, M. Nowak, and W. Sładek, "System-level hardening techniques used in the COTS-based data processing unit," in *2nd European Workshop on On-Board Data Processing (OBDP2021)*, 2021.

[19] N.-J. Wessman, F. Malatesta, S. Ribes, J. Andersson, A. García-Vilanova, M. Masmano, V. Nicolau, P. Gomez, J. L. Rhun, S. Alcaide, G. Cabo, F. Bas, P. Benedicte, F. Mazzocchetti, and J. Abella, "De-risc: A complete risc-v based space-grade platform," in *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2022, pp. 802–807.

[20] P. Nannipieri, G. Giuffrida, L. Diana, S. Panicacci, L. Zulberti, L. Fanucci, H. G. M. Hernandez, and M. Hubner, "ICU4SAT: A general-purpose reconfigurable instrument control unit based on open source components," in *2022 IEEE Aerospace Conference*, 2022, pp. 1–9.

[21] V. Leon, E. A. Papatheofanous, G. Lentaris, C. Bezaitis, N. Mastorakis, G. Bampilis, D. Reisis, and D. Soudris, "Combining fault tolerance techniques and COTS SoC accelerators for payload processing in space," in *2022 IFIP/IEEE 30th International Conference on Very Large Scale Integration (VLSI-SoC)*, 2022.

[22] F. Bubenhagen, B. Fiethe, T. Lange, H. Michalik, and H. Michel, "Reconfigurable platforms for data processing on scientific space instruments," in *2013 NASA/ESA Conference on Adaptive Hardware and Systems (AHS-2013)*, 2013, pp. 63–70.

[23] *CompactPCI Serial Space Specification*, PIMCG Std., 2017.

[24] J. Hauslage, M. Lebert, and H. Müller, "Eu:CROPIS – euglena and combined regenerative organic-food production in space," in *Life in Space for Life on Earth (Joint Life Sciences Meeting of ISGP, ESA and CSA)*, Juni 2014.

[25] T. Gärtner, C. J. Treudler, F. Dannemann, and M. Jetzschmann, "Scalable avionics for the dlr micro- and minisatellite platforms s2tep and compsat," in *DASIA – Data Systems In Aerospace*, 2017.

[26] T. Peng, B. Weps, K. Höflinger, K. Borchers, D. Lüdtke, and A. Gerndt, "A new spacewire protocol for reconfigurable distributed on-board computers," in *International SpaceWire Conference, Yokohama, Japan, October 25-27, 2016*. IEEE, October 2016, pp. 175–182.

[27] Z. A. H. Hammadeh, T. Franz, O. Maibaum, A. Gerndt, and D. Lüdtke, "Event-driven multithreading execution platform for real-time on-board software systems," in *15th Workshop on Operating Systems Platforms for Embedded Real-Time applications (OSPERT), Stuttgart, Germany, July 9, 2019*, A. Lackorzynski and D. Lohmann, Eds., 2019, pp. 29–34. [Online]. Available: https://elib.dlr.de/128249/

[28] DLR. (2022) Open modUlar sofTware PlatfOrm for SpacecrafT. [Online]. Available: https://www.github.com/dlr-ry/outpost-core

[29] S. Ulamec, P. Michel, M. Grott, U. Böttger, S. Schröder, H.-W. Hübers, Y. Cho, F. Rull, N. Murdoch, P. Vernazza, J. Biele, S. Tardivel, and H. Miyamoto, "Science objectives of the MMX rover," in *73rd International Astronautical Congress (IAC)*, September 2022.

[30] A. Kovalov, T. Franz, H. Watolla, V. Vishav, A. Gerndt, and D. Lüdtke, "Model-based reconfiguration planning for a distributed on-board computer," in *12th System Analysis and Modelling (SAM) Conference - Languages, Methods and Tools for AI-based Systems, co-located with MODELS 2020, Virtual Event, Oct. 19-20, 2020*. Association for Computing Machinery (ACM), October 2020, pp. 55–62.

[31] J. v. Kistowski, J. A. Arnold, K. Huppler, K.-D. Lange, J. L. Henning, and P. Cao, "How to build a benchmark," in *Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering*, ser. ICPE '15. New York, NY, USA: Association for Computing Machinery, Jan. 2015, pp. 333–336.

[32] M. M. Elbarrawy, "Performance evaluation for a distributed on-board computer," Master's thesis, Deggendorf Institute of Technology, January 2023. [Online]. Available: https://elib.dlr.de/194056/

[33] J. J. Dongarra, P. Luszczek, and A. Petitet, "The LINPACK benchmark: past, present and future," *Concurrency and Computation: Practice and Experience*, vol. 15, no. 9, pp. 803–820, Aug. 2003.

[34] R. Weicker, "An overview of common benchmarks," *Computer*, vol. 23, no. 12, pp. 65–75, Dec. 1990.

[35] D. Steenari, L. Kosmidis, I. Rodriguez-Ferrandez, A. Jover-Alvarez, and K. Förster, "OBPMark (on-board processing benchmarks) – open source computational performance benchmarks for space applications," in *2nd European Workshop on On-Board Data Processing (OBDP2021)*, Jun. 2021, publisher: Zenodo Version Number: 1.0.

[36] *OBPMark (On-Board Processing Benchmarks)*, 2022. [Online]. Available: https://github.com/OBPMark/OBPMark/wiki

[37] D. Evans and M. Merri, "OPS-SAT: A ESA nanosatellite for accelerating innovation in satellite control," in *SpaceOps 2014 Conference*, 2014, p. 1702.

[38] M. Sandler, A. G. Howard, M. Zhu, A. Zhmoginov, and L. Chen, "Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation," *CoRR*, vol. abs/1801.04381, 2018. [Online]. Available: http://arxiv.org/abs/1801.04381