Received March 6, 2020, accepted March 22, 2020, date of publication March 25, 2020, date of current version April 8, 2020.

*Digital Object Identifier 10.1109/ACCESS.2020.2983308*

# Run-Time Reconfigurable MPSoC-Based On-Board Processor for Vision-Based Space Navigation

**ARTURO PÉREZ[1], ALFONSO RODRÍGUEZ[1], (Student Member, IEEE), ANDRÉS OTERO[1], (Member, IEEE), DAVID GONZÁLEZ ARJONA[2], ÁLVARO JIMÉNEZ-PERALO[2], MIGUEL ÁNGEL VERDUGO[2], AND EDUARDO DE LA TORRE[1]**

[1]Centro de Electrónica Industrial, Universidad Politécnica de Madrid, 28006 Madrid, Spain
[2]GMV Aerospace and Defence SAU, 28760 Madrid, Spain

Corresponding author: Eduardo De La Torre (eduardo.delatorre@upm.es)

This work was supported in part by the ENABLE-S3 Project through the ECSEL Joint Undertaking under Grant 692455, and in part by the European Union's Horizon 2020 Research and Innovation Program and Austria, Denmark, Germany, Finland, Czech Republic, Italy, Spain, Portugal, Poland, Ireland, Belgium, France, The Netherlands, U.K., Slovakia, and Norway.

**ABSTRACT** This paper describes a reconfigurable architecture for an on-board processor to be used in space exploration critical systems. It relies on, a dynamically reconfigurable multi-accelerator hardware architecture that provides transparent reconfiguration and scalable performance, dependability, and power consumption, at run-time. The architecture is integrated under an RTEMS operating system, which manages reconfiguration and fault mitigation in a fully-compatible way with space requirements. In this work, the proposed processor is used to implement a vision-based navigation system, providing fully autonomous adaptation to different phases of a space mission timeline or events that the spacecraft may encounter during its lifespan. Results show that reconfigurability enables the practical usage of Commercial Off-The-Shelf (COTS) Multiprocessor Systems-on-Chips (MPSoCs) in real scenarios.

**INDEX TERMS** Vision-based navigation, reconfigurable on-board processor, high performance embedded computing, real-time operating systems.

## I. INTRODUCTION

Small bodies, including asteroids and comets, have been and will continue being targets of different in-situ exploration and sample-return space missions since they constitute an unrivaled source of information about the history of the Solar System. A significant example is *Rosetta*, which was designed and managed by the European Space Agency (ESA), having Comet 67P/Churyumov-Gerasimenko as its destination [1].

Reaching and landing on an asteroid requires that tight orbital and descent specifications are followed. This goal is only possible with extremely accurate Guidance, Navigation, and Control (GNC) systems. The accuracy required not to compromise the success of the mission cannot be guaranteed using only inertial units. Inertial measurements diverge quickly, and thus, they have to be fused periodically with absolute references. The use of cameras is the only solution to provide the absolute reference since, in the far and medium distance ranges from the spacecraft to the target, it is not possible to relay on altimeter sensors. The extraction of navigation data from images is called Vision-Based Navigation (VBN).

VBN algorithms must necessarily run autonomously on board the spacecraft since the communication delay with the ground control center would prevent remote processing. Taking the NASA *InSight* mission to Mars as an example, the Entry, Descent, and Landing (EDL) phase until reaching the surface of Mars lasted six minutes [2], while a one-way communication from Earth to the spacecraft required about ten minutes, in best cases scenarios. However, on-board VBN constitutes a challenge, since it involves the execution of complex image processing algorithms at a frame rate high enough to guarantee the convergence of the navigation Kalman filters used for multi-sensor data fusion [3].

The associate editor coordinating the review of this manuscript and approving it for publication was Venkata Ratnam Devanaboyina.

These requirements surpass the performance space-qualified processors, such as PowerPC-FT or LEON-based cores, can provide, forcing the use of ad-hoc hardware-accelerated solutions, typically in the form of ASICs [4]. For these reasons, VBN has been identified as one of the applications requiring high-performance embedded computing in space. From this perspective, Lentaris *et al.* provide in [5] an extensive evaluation of multiple computing platforms specific for VBN systems, concluding that new-generation space-grade CPUs are still one order of magnitude slower than what is needed for reliable autonomous VBN.

In the last years, we are witnessing a paradigm change in the space industry. The *NewSpace* approach is making agencies and companies to be interested in cost-effective solutions, as described in [6]. Thus, FPGAs are increasingly used in space to substitute the unaffordable development of ASICs. Different rad-hard FPGA devices are now accepted and widely used in commercial missions. Most of these rad-hard devices are based on One-Time Programmable (OTP) or flash technologies for the device configuration memory, such as the Microsemi RTG4, which hinders the exploitation of in-flight reconfiguration. Moreover, their performance is much lower than Commercial Off-the-Shelf (COTS) counterparts [4], which makes them unsuitable for VBN. Only the SRAM-based rad-hard Xilinx Virtex-5QV FPGA or the emerging BRAVE FPGAs from Nanoexplore can provide the high-performance and the number of logic resources required by these applications but at a cost much higher than COTS non-radiation-hardened versions.

Besides combining low cost with state-of-the-art processing performance, commercial SRAM-based reconfigurable devices offer the possibility of adapting the device logic at run-time depending on the different stages the mission may encounter during its lifetime. This way, a single processing unit can be reused for multiple non-concurrent payload processing tasks, reducing mass, volume, and energy, which in the end also translates into cost reduction and simplifies overall avionics system design. In spite of these clear benefits, COTS SRAM-based FPGAs are more susceptible to radiation-induced faults in their configuration memory, so they require the application of specific hardening techniques to protect not only the application logic but also the configuration memory. Going one step forward FPGAs, state-of-the-art Multiprocessor Systems-on-Chips (MPSoCs) with programmable logic allow combining the benefits of both software multiprocessing and reconfigurable hardware in a single device.

The main contribution of this paper is to provide a new cost-efficient software/hardware On-Board Processor (OBP) architecture that effectively exploits Dynamic and Partial Reconfiguration (DPR) on COTS reconfigurable MPSoCs in space applications. This proposal is based on the ARTICo$^3$ multi-accelerator architecture [7], which provides transparent reconfiguration for performance scalability and dependability, at run-time. The adaptability provided by the architecture is not only functional, but it has also been extended to contribute to the system fault-tolerance. Besides the architecture, ARTICo$^3$-based solutions provide a run-time software library to manage the reconfigurable accelerators. In this work, the ARTICo$^3$ runtime has been integrated with RTEMS, a real-time operating system certified and compliant with space requirements. Application-independent *support tasks* in charge of providing system-level reconfiguration and Fault Detection, Isolation, and Recovery (FDIR) features, have also been integrated into this scheme, together with application-specific *mission tasks*. The proposed mixed software/hardware architecture has been tested for VBN applications, allocating three different vision-based processing algorithms that could not otherwise fit in one single device. Each algorithm will be used at a different stage depending on the distance to the target and the mission phase. The proposed architecture has been implemented and evaluated on a Zynq UltraScale+ MPSoC device.

The rest of the paper is organized as follows. In Section II, the envisaged mission scenario is provided as motivation. In Section III, the state-of-the-art in the use of reconfigurable FPGAs for space applications is summarized. A background with the most significant technologies for this work is included in Section IV. The proposed system architecture, the hardware subsystem management, and fault-mitigation techniques are described in Sections V, VI, and VII. The mapping of the vision-based navigation application on the proposed platform is discussed in VIII. Finally, experimental results are offered in Section IX, and conclusions and future work are summarized in Section X.

## II. USE CASE SCENARIO

The scenario envisaged as the use case for the proposed on-board processor entails a spacecraft lander carrying on a rover to be deployed on an astronomical body. Depending on the relative size of the target under the vision of the spacecraft camera, different magnitudes can be extracted from the images. Therefore, the most appropriate navigation algorithms for each particular time depend on the distance from the lander to the target.

During the navigation phase to the celestial body, the spacecraft lander makes use of the camera installed in the rover for the descent and landing part using an *absolute navigation* image processing interchanged by fast hardware reconfiguration with a *relative navigation* image processing implementation in the FPGA. The absolute navigation technique relies on surface features detection and matching of a priori extracted landmarks. This technique is slower than relative navigation techniques, and for that reason, it is devoted to execution at higher altitudes right before the final descent.

During the final descent, the technique utilized is the *relative navigation* based on surface feature detection and tracking among consecutive frames. This technique is implemented for fast execution in order to avoid measurement dispersion, to accommodate the fast dynamics of the spacecraft and the quick reaction capabilities needed to control the safe landing. After landing, once that the spacecraft probe is on the ground, the FPGA is once again reconfigured for surface operations to host a solution based on stereo vision disparity Semi-Global Matching (SGM) in the same FPGA.

Therefore, there is a need for adapting the navigation algorithm at run-time, depending on the operational stage the spacecraft is. In the current exploration and landing missions, computing architectures with at least three FPGAs would be needed to allocate these three algorithms. The proposal of this work shows an avionics architecture that allows reconfiguring a single FPGA device, simplifying the implementation and reducing mass and power, while providing a common platform that may be used over the different phases of the mission. Absolute, relative, and stereo-vision navigation algorithms can be interchanged using fast hardware reconfiguration. This way, the spacecraft lander will use the same FPGA for the different navigation algorithms. Since these are critical safety operations, the re-programming time in which the FPGA is not operative constitutes an essential factor and one of the performance indicators.

## III. RELATED WORK

A selection of relevant works in the state-of-the-art showing the usage of rad-hard FPGAs in space and, in particular, COTS-based solutions relying on dynamic and partial reconfiguration, are described in the three following sections.

### A. RAD-HARD SRAM-BASED FPGAs IN SPACE MISSIONS

Different space missions have been launched since the beginning of the century, relying on SRAM-based FPGAs as the leading processing technology.

The Cibola Flight Experiment (CFE) [8], developed by Los Alamos National Laboratory and launched in 2007, is one of the most relevant examples. Its goal was actually to evaluate the feasibility of using SRAM-based FPGAs for on-board processing. The payload was managed by a rad-hard processor, while three reconfigurable computer modules were used for experimentation. Each module was composed of three Xilinx Virtex XQVR1000 for data processing and one radiation-tolerant Actel RT54SX32S, providing watchdog monitoring and a configuration interface to the FPGAs. Several experiments were carried on the CFE, ranging from the characterization of how FPGAs are affected by radiation to the evaluation of different mitigation techniques for radiation effects.

SRAM-based FPGAs have also been employed for critical functions in exploration missions. One example is the lander used to deploy the Spirit and Discovery rovers on the Mars surface, promoted by the Jet Propulsion Laboratory [9]. In these missions, Xilinx XQR4062XL FPGAs were in charge of controlling the descending and landing operations. Triple Modular Redundancy at design level and redundant FPGAs at component level were employed, coupled with full FPGA reconfiguration (i.e., scrubbing), as fault mitigation techniques. Component redundancy is a robust approach, but it suffers from high costs in terms of weight, price, and energy.

Another interesting example is the Fraunhofer On-Board Processor (FOBP) [10], proposed by the Fraunhofer Institute for Integrated Circuits, as a flexible communication payload for satellite-based digital signal processing applications. This development will be part of the scientific payload of the Heinrich Hertz communication satellite planned to be launched in 2021. The FOBP consists of two radiation-hardened Virtex-5QV FPGAs that can be entirely reconfigured from the ground station. That way, the processor can be adapted to new communication standards and to changing environmental conditions at any time while in orbit. Thus, the FOBP can perform signal processing at the physical layer (including re-modulation or re-encoding), usually not performed on board, since the lack of flexibility of existing non-reconfigurable satellite communication systems would prevent the adoption of new standards [11]. Authors in [12] report an evolution of the FOBP. This work raises the possibility of using non-hardened devices, such as the Kintex-UltraScale XCKU040 or Zynq-UltraScale+ ZU19EG, as part of the FOBP for commercial Low Earth Orbit (LEO) communication applications.

### B. COTS MPSoCs IN SPACE MISSIONS

In the field of reconfigurable devices, special attention is being paid on reconfigurable MPSoCs. These devices embed in the same integrated circuit multiprocessors, SRAM-based FPGAs, and dedicated peripherals. The flexibility offered by MPSoCs cannot be found in other devices due to the combination of the inherent flexibility of software and the one achievable with reconfigurable circuits. This heterogeneity can be further exploited to harden the system against radiation thanks to the diversity provided by pairing both architectures on a single die. These considerations were taken into account to develop the CHREC Space Computer (CSP) [13], an onboard computer implemented mixing radiation-hardened and COTS components. This computer has high computing capabilities due to being based on a Zynq platform from Xilinx. Software and hardware hardened techniques have been employed to maximize the global design reliability, which has been validated with in-flight data obtained in the context of the STP-H5 mission. The CSP can be hardened with a DPR-compatible scrubber based on updating the golden copy when a new configuration is loaded into the FPGA [14]. Similar to the CSP, the APEX-SoC solution [15] leverages the resources offered by a single Zynq device to implement instrument data acquisition and processing stages. They applied several hardening techniques to protect the logic of the design implemented in the FPGA, the application data, and the processors. The hardening methods are mainly based on redundancy, watchdogs, ECC codes, and scrubbers.

The benefits of reconfigurable MPSoCs have also attracted the attention of ESA, which is developing the OPS-SAT program [16]. The main goal of this initiative is to develop a satellite using COTS wherever possible, avoiding new developments. An ALTERA Cyclone V device will be the primary computing platform employed in the payload, so it is flexible enough to allow the replacement of applications while in flight.

The NINANO platform (Numeric Intensive Node for Applications mono-module adaptation for nanosatellites) [17] is an OBP based on a Xilinx Zynq-7030. This on-board processor will be used in at least two scientific missions: the EYE-SAT [18] and the SERB-SAT [19]. The EYE-SAT mission has two main scientific objectives: to study the intensity and polarization of the zodiacal light over a vast portion of the sky, as well as to provide a 360° colored picture of the Milky Way. SERB-SAT will be used to measure the total solar irradiance to understand how this magnitude affects climate.

### C. DYNAMIC AND PARTIAL RECONFIGURATION IN SPACE APPLICATIONS

The use of in-flight reconfigurable devices in avionics is gaining the attention of companies and space agencies. First in-flight reconfiguration was reported by [20] in the context of the FedSat mission [21]. More recently, in the already mentioned Cibola flight experiment, reconfiguration was used for correcting Single Event Upsets (SEU), as well as to adapt the functionality in the FPGA to the requirements of different experiments. Exploiting this dual-purpose reconfiguration is common when applying this technique in radiation prone environments.

Regarding fault mitigation, it must be mentioned the work done in [22], where authors combine the built-in features for SEU mitigation embedded in Xilinx Zynq MPSoCs with a readback scrubber. With this approach, a global mitigation strategy is designed taking advantage of the fast detection and correction capabilities of information redundancy coding based scrubbers, complemented with readback-based scrubbers. This allows repairing any fault detected in the configuration memory. Another interesting work is the one presented in [23], where authors combine DPR with configuration memory scrubbing. They propose a solution based on a golden copy. Periodically, redundancy codes are computed and compared from parts of both the configuration memory and the golden copy. To perform DPR, first, they modify the appropriate section of the golden copy with the new configuration data. When the scrubbing process passes through the new data of the configuration memory and detects differences between it and the golden copy, the corrected configuration is reloaded in the system. Contrary to this approach and the previously mentioned from [14], we present in this paper a DPR-aware compatible scrubber that does not require to generate or modify the golden copy every time a reconfiguration is performed.

The work developed by ESA in [24] is very remarkable since it demonstrates the effectiveness of reconfiguration in the new FPGA platform BRAVE, developed by NanoExplore. Differently to previous works, BRAVE FPGAs are space-grade devices that count on a rad-hard fabric. Another relevant feature of this platform is the possibility of using a SpaceWire interface for modifying the configuration memory. This new approach was evaluated in [25], where the authors compare the reconfiguration times when either SpaceWire or JTAG interfaces are employed.

## IV. BACKGROUND INFORMATION

Before providing details on the proposed reconfigurable architecture, relevant context information is provided in this section.

### A. ZYNQ UltraScale+ MPSoC ARCHITECTURE

Zynq UltraScale+ is the latest generation of Xilinx MPSoCs. It is manufactured in TSMC FinFET 16nm technology, and it features an ARM-based processing system. It has been selected as the technology underneath in this work due to the experience Xilinx has on SRAM-based reconfigurable devices, which results in sophisticated tools and IPs. Xilinx also accumulates a relevant flight experience with the space-grade Virtex-5QV, while their non-hardened devices are also attracting the interest of the space industry. It must also be mentioned the device characterization effort carried out by the Xilinx Radiation Test Consortium (XRTC) that, together with other industrial and academic partners, is carrying out the first experiments on the SEU characterization of the Zynq UltraScale+ [26], [27]. Moreover, different initiatives have been carried out by the industry to provide radiation-hardened versions of the ARM processors included in the Zynq Ultrascale+ [28], [29]. Therefore, the proposals developed in this work could be implemented in the future using full rad-hard devices.

The Zynq UltraScale+ platform offers two different hardwired processor fabrics: the Application Processing Unit (APU) and the Real-Time Processing Unit (RPU). The APU is composed of up to four application-oriented Cortex-A53 processors, running up to 1.3 GHz, while the RPU consist of two real-time oriented Cortex-R5 processors, working up to 533 MHz. In some parts of the family portfolio, an ARM Mali 400 Graphical Processing Unit (GPU) is also included. As a whole, the APU and the RPU constitute the Processing System (PS). The PS is combined with an FPGA. The FPGA is the reconfigurable part of the chip and is known as the Programmable Logic (PL). The circuitry implemented on the PL can be (re-)configured from a processor in the PS (using the PCAP interface), the FPGA itself (using the ICAP interface), or an external device. Further details on the device reconfiguration can be found in the Xilinx UltraScale Architecture Configuration User Guide [30].

### B. ARTICo$^3$ ARCHITECTURE

The ARTICo$^3$ [7] is a hardware-based processing architecture for high-performance embedded reconfigurable computing.

As shown in Figure 1, the hardware architecture in ARTICo$^3$ is mainly divided into a static part (i.e., the infrastructure that delivers data to the different accelerators) and

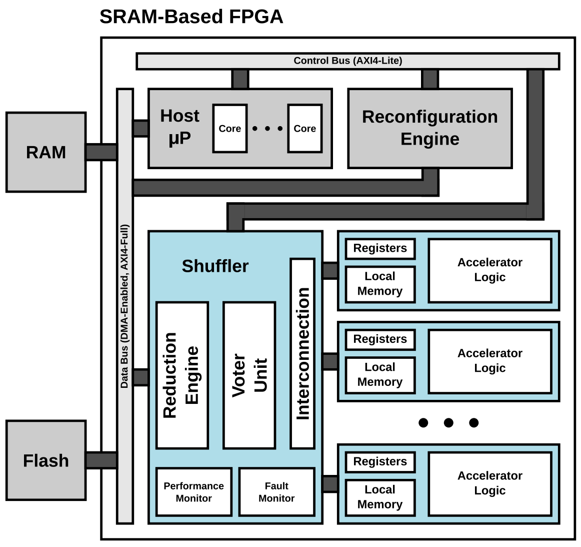

**FIGURE 1.** ARTICo$^3$ architecture, including the Host microprocessor, the shuffler and the dynamically reconfigurable logic for the accelerators.

a dynamic part (the hardware accelerators). The ARTICo$^3$ architecture features a DMA-capable datapath, whose internal structure can be configured at run-time to support different processing profiles. The reconfigurable datapath belongs to the static part, which is automatically loaded during the system startup. The dynamic part is divided into several reconfigurable partitions (or slots), where the hardware accelerators are loaded using DPR when the user requires it.

Using dynamic and partial reconfiguration as its technological foundation, ARTICo$^3$ allows a dynamic exploration of the space of possible solutions, changing the working trade-offs at run-time. Hence, performance-oriented or fault-tolerant processing can be selected on-demand.

In performance-oriented operation points, multiple instances of a given hardware accelerator are loaded in the FPGA fabric. The ARTICo$^3$ datapath is then configured to send different input data to each instance. The parallel instances of the same hardware accelerator are then executed simultaneously on different data, in a SIMD-like fashion.

From the perspective of dependability, ARTICo$^3$ provides basic structures to support selectable fault tolerance and fault detection. Fault-tolerant execution is achieved using the logic that acts as a gateway between the static region and the reconfigurable partitions. Thus, in fault-tolerant profiles, the datapath is configured to send the same data simultaneously to two or three accelerators and gathered through a configurable voting unit to implement Dual Modular Redundancy (DMR) or Triple Modular Redundancy (TMR) execution modes. Moreover, this fault-tolerant mechanism is performed on a datum basis by taking advantage of the bus-based communication infrastructure available in ARTICo$^3$ based systems. In turn, fault detection is supported by a lightweight monitoring infrastructure, where each reconfigurable partition has an associated error counter. Control signals coming from the configurable voting unit are in charge of increasing the corresponding counters whenever different results are obtained in one execution. The embedded monitoring infrastructure in ARTICo$^3$ allows users to analyze system performance and error rates.

### C. VISION-BASED NAVIGATION ALGORITHMS

As motivated in Section II, three different space environment scenarios have been identified: absolute navigation based on landmark matching, relative navigation based on feature tracking, and stereo vision algorithms for close operation.

Absolute navigation uses image processing techniques to find relevant features corresponding to remote stellar bodies in the image collected by the navigation camera. Features are then mapped and compared with a database available on board and previously generated. The mapping of multiple features allows the navigation filter to derive an accurate position estimation related to the target surface. Absolute navigation requires an image processing algorithm that provides edges that are later post-processed to extract landmarks of the surface. The algorithm selected in this work is the widely known *Canny edge detector*.

In relative navigation, the image processing part of the algorithm is responsible for the extraction of the landmark points in the image and tracking them. However, it does not rely on a thorough comparison with a known database, but on the relative tracking of the displacement of the detected features from one frame to the subsequent one. Following the displacement of up to 100 distinct points observed in the images, the navigation filter can accurately calculate the position of the spacecraft. The *Harris Corner* and *Minimum Eigen Value corner* algorithms are used to find corners, to be used as useful features to track.

In turn, to obtain 3D information regarding the target body, a stereo vision algorithm working on information from two separate cameras is used. The system performs the extraction of lines that are matched in the two stereo views; these matches are then used to estimate pose. The selected algorithm to be accelerated in hardware is the semi-global matching algorithm [31].

Finally, the fusion of the data produced by inertial sensors with the absolute references obtained by the three proposed vision-based navigation algorithms is carried out by a standard Kalman Filter, similar to those reported in [3].

## V. RECONFIGURABLE ON-BOARD PROCESSOR ARCHITECTURE

This section describes the proposed hardware and software architecture for the reconfigurable on-board processor, including further details on platform adaptation and the proposed templates for software tasks.

### A. HARDWARE ARCHITECTURE

The hardware architecture proposed for the on-board processor is composed of three subcomponents: the Real-Time Processing Unit, the ARTICo$^3$ infrastructure, and the Xilinx Soft Error Mitigation (SEM) IP [32]. A simplified representation of the proposed on-board processor is shown in Figure 2.

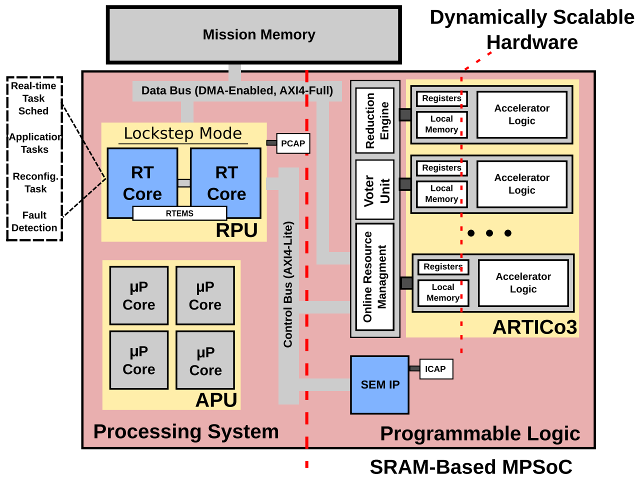

FIGURE 2. Block diagram of the baseline architecture of the proposed Reconfigurable On-board Processor for space applications.

The RPU has been selected as the leading software computing unit since the Cortex-R5 processors that make up this unit can run in lock-step mode. In other words, they can be configured within a physical and temporal DMR scheme in which the two processors are executing two exact copies of the same application code, but with a delay of some clock cycles [33]. If a mismatch between the results they provide is detected, the whole computation is canceled and repeated. The PCAP and the ICAP are the interfaces for accessing the configuration memory. They are used by the RPU and the SEM-IP, respectively. The RPU uses the PCAP to adapt the set of hardware accelerators to new mission stages, to alternate between ARTICo$^3$ profiles (see section IV-B) and, to perform active repairs using scrubbers. The SEM-IP [34] requires the use of the ICAP and provides an Error Redundancy Coding based scrubber that offers low detection time. The information about the scrubbers implemented in the platform is further explained in section VII. Data exchanges are done via an AXI4-Full DMA-capable bus, while control operations (setting up registers for accelerators profile selection, SEM-IP control) is performed via an AXI4-Lite bus.

Only the hardware accelerators loaded in the ARTICo$^3$ architecture are dependent on the mission goals, while the rest of the modules are application-independent. As a result, and thanks to its hardware reconfiguration capabilities, the reconfigurable on-board processor constitutes a generic solution for space applications, providing mission independent hardware and software elements that can be integrated with mission-specific ARTICo$^3$ accelerators and software application tasks.

When new applications have to be supported, hardware accelerators must be made compliant with the ARTICo$^3$ architecture. The ARTICo$^3$ toolchain provides architectural templates for the interfaces, and automated scripts to produce the required FPGA configuration files (i.e., bitstreams). These bitstreams can be used at run-time to change the hardware accelerators available in each reconfigurable slot.

## B. SOFTWARE ARCHITECTURE

The use of a Real-Time Operating System (RTOS) is crucial in on-board processors to guarantee they comply with the requirements of the space domain. In this work, RTEMS (Real-Time Executive for Multiprocessor Systems) has been selected with this aim. RTEMS is a multi-threaded hard real-time OS widely used in different NASA and ESA missions. A custom RTEMS Board Support Package (BSP) has been developed in this work to make it compatible with the Zynq UltraScale+ MPSoC board used in the experimental setup. Key aspects addressed when porting RTEMS to this platform have been the initialization of the memory protection unit of the processors, the development of the clock driver, and the management of the booting stage. Further low-level support was added to the BSP by re-using the existing bare-metal drivers for the different peripherals on the board.

The real-time OS is in charge of scheduling and coordinating the execution of software tasks while guaranteeing real-time constraints and deterministic behavior. RTEMS is also in charge of arbitrating access to shared resources in the device, such as the reconfiguration port. Therefore, the RPU featured with RTEMS constitutes the foundation upon which mission applications and support tasks have been implemented. The former includes the control-intensive sections of the mission-specific navigation algorithms that are not susceptible to being accelerated in hardware. The latter, on the other hand, includes the FDIR functionalities, the reconfiguration functions included in the ARTICo$^3$ runtime, and the communication data-handling. All these tasks qualify as platform support tasks since they allow applications to exploit the main features offered by the on-board processor, as well as to solve the problems associated with using reconfigurable devices in radiation prone environments. Of particular interest is the data handler task, which processes messages coming from an external mission controller through the command and control data-link. Those messages are capable of producing a change in the phase of the mission, therefore triggering in-flight reconfiguration.

The software runtime of the ARTICo$^3$ architecture, which was initially provided as a Linux driver and the corresponding user-space software library, was also ported to RTEMS as part of this work. This was performed considering the tight constraints real-time applications must comply with, being determinism one of the main concerns. In the proposed RTEMS-based implementation, the execution flow of each kernel is controlled by a specific thread that must be created by the user for this aim. These threads enter a *sleep state* after commanding a DMA transaction to the accelerators or while they are waiting for the hardware accelerators to finish a computation. As a result, the processor is free for other threads to execute. This feature, combined with the fact that all the API routines are thread-safe, ensures that different kernels can be used concurrently in a single application. Particular effort was put to avoid potential data-race situations with coordinated access to shared data.

**TABLE 1.** UC scenario modes and APs involved in each mode.

| | APs | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **Modes** | Data-Handling | Fault-Detection | Reconfiguration | GNC | Relnav | Relnav 2 | Absnav | Stereo-Vision |
| Coasting | x | x | | | | | | |
| Navigation Init | x | x | | x | | | | |
| Absolute Navigation | x | x | | x | | | x | |
| Braking Phase Nav | x | x | | x | x | x | x | |
| Relative Navigation | x | x | | x | x | x | | |
| Terminal Descent | x | x | | x | | | | |
| Landing Operations | x | x | | | | | | x |
| Repair Mode | x | | x | x | | | | |
| Reconfiguration Mode | x | | x | x | | | | |

**Algorithm 1** AP Template

```
thread AP_<NAME>_THREAD {
rtems_rate_monotonic_create(AP_<name>_period)
loop:
    rtems_rate_monotonic_period(AP_<name>_period)
    function AP_<name>_receive() {
        if new_messages == TRUE then
            parse_messages()
        end if
    end function
    function AP_<name>_step() {
        if state == AP_ < name > _ON_state then
            perform_operations()
        end if
    end function
    function AP_<name>_publish() {
        if new_output_data == TRUE then
            publish_data()
        end if
    end function
    goto loop.
}
end thread
```

## C. MISSION AND SUPPORT TASK TEMPLATES

A software task, referred to as Application Processing (AP), accompanies each hardware kernel included in the on-board processor. The AP includes the software partition of the algorithm, the control of the execution of the corresponding ARTICo$^3$ hardware accelerators, and the data supply to these accelerators. The template proposed in this paper for the APs is represented in Algorithm 1. This template is not exclusive for the processing tasks requiring hardware acceleration. Purely software tasks are also created according to this structure, facilitating message passing among all the tasks in the system.

All the APs are executed periodically using a *rate monotonic* [35] scheduling under the RTEMS OS. As can be seen in Algorithm 1, every time an AP is executed it does three things: (i) parse new messages (if any), (ii) perform operations (if the task is currently active), and (iii) deliver output data (if any). New applications with hardware acceleration requirements can be ported to the on-board processor by adapting the hardware accelerators to the ARTICo$^3$ accelerator template and developing the software counterparts following the standardized template and the ARTICo$^3$ runtime API.

As has been already explained, support tasks also follow the AP structure provided in Algorithm 1, and they constitute the mission independent software elements of the OBP that is always present in the system regardless of the mission of the spacecraft. The specific support APs included in the platform are the following:

- *Data-Handling AP*: Processes the messages from the command and control data-link, which can trigger the adaptation of the platform.
- *Fault-Detection AP*: Performs the platform fault detection, using error redundancy coding and readback scrubbing techniques.
- *Reconfiguration AP*: Centralizes all the hardware reconfiguration requests for both functional adaptation and fault recovery.

All these support APs are described in detail in the next sections.

## D. MODE DEFINITION FOR PLATFORM ADAPTATION

Platform in-flight adaptability is achieved by the definition of different execution modes, which differ in the set of enabled tasks. Application-independent support modes for executing the mode change as well as for fault repair are always present in the platform.

The *Reconfiguration Mode* is the one in charge of executing the mode change, and thus, the platform always enters this mode when switching between any other two modes. An on-ground mission controller triggers the mode change, using the command and control data-link. In turn, the system enters the *Repair Mode* when the *Fault-Detection AP* detects a system fault. In the *Repair Mode*, the DPR capabilities of the MPSoC are used to recover the initial state of the configuration memory.

Apart from support modes, many others can be defined according to the application requirements. For the vision-based application presented in this work, table 1 gathers all the implemented modes, as explained next. During the *coasting* stage, the autonomous GNC function is under hibernation, so neither the GNC software nor the FPGA accelerators are activated. Before reaching the asteroid close operations, there is a mode change entering in *navigation init* where the GNC software is awakened for the initialization of units, the configuration of software functions, and sensors. No FPGA accelerators are yet activated. At some mission point, the Earth's ground operations initialize the on-board autonomous navigation, passing to the *absolute navigation* mode. Between absolute and relative navigation modes, an intermediate *braking mode* has been defined, in which the spacecraft initiates descent operations. During this phase, both absolute navigation and relative navigation techniques co-exist in an interleaved manner to allow the proper relative navigation initialization and a smooth mode change. Afterward, the *relative navigation* technique is executed until the very end of the landing over the surface. Relative navigation is based on two image processing techniques. First feature detection is carried out, followed by feature tracking. Therefore there are two APs, named here *Relnav* and *Relnav_2*, associated with this mode. Before touch down, there is the last mode change to *terminal descent*, where spacecraft follows a free fall with no spacecraft control to avoid surface contamination due to thrusting. Finally, after the landing, the rover vehicle starts the *landing operations* over the celestial body surface, making use of the same avionics, so the image processing technique of stereo vision for terrain navigation is activated.

The next section provides further details on how the hardware subsystem is managed on the platform.

## VI. HARDWARE SUBSYSTEM MANAGEMENT

This section focuses on the main elements involved in the management of the hardware subsystem. First, the *Reconfiguration* and the *Data-Handling APs*, which control mode changes, are described. Then, the integration of ARTICo$^3$ primitives in the mission-specific APs is detailed.

### A. DATA-HANDLING AP AND RECONFIGURATION AP FOR MODE CHANGES

The *reconfiguration AP* orchestrates all the tasks performed in the on-board processor related to hardware reconfiguration. Reconfiguration can be triggered either to repair SEUs in the configuration memory of the FPGA or to execute a mode change by reconfiguring the hardware accelerators in the FPGA logic. Hence, this AP reacts to system demands in the form of requests which can be triggered by the *Fault-Detection AP* or by the *Data-Handling AP*. By using a centralized scheme, accesses to the PCAP configuration interface are resolved by an arbitration entity. This section focuses on functional adaptation. The use of reconfiguration for fault tolerance is described in section VII.

The *Data-Handling AP* transforms the mode change requests received from an external mission controller into reconfiguration requests. Mode changes cannot be

```
void AP_Data_Handling_step
{
  ... // Some flag checks
  if ( current_mode != requested_mode ) {
     if ( current_mode != on_mode_change_m ) {
       switch ( requested_mode ) {
       mode A:
         set_flag_reconfig_required( true );
         artico_load_req("kernel_A", 1, ...);
         artico_load_req("kernel_A", 2, ...);
         artico_load_req("kernel_A", 3, ...);
         ...
         break;
       mode B:
         set_flag_reconfig_required( true );
         artico_load_req("kernel_B", 1, ...);
         artico_load_req("kernel_C", 2, ...);
         ...
         break;
       mode C:
         set_flag_reconfig_required( false );

     }
     current_mode = on_mode_change_m;
  } else {  // current_mode == on_mode_change_mode
    if ( get_flag_reconfig_required == false )
      current_mode = requested_mode;
  }
  ...
}
```

**Code. 1.** Mode changes implementation.

immediately performed because the APs using hardware acceleration cannot be stopped abruptly, in the middle of a data transaction to the hardware accelerators. In other words, the PL must be adapted to the next mode before it starts. For this reason, the *Data-Handling AP* enqueues reconfiguration requests into a buffer shared with the *reconfiguration AP*. Requests are issued by calling the `artico3_load_req` function. Code 1 shows the implementation scheme of the *Data-Handling AP*. Mode change requests are stored in the variable `requested_mode` which is compared with the `current_mode` variable whenever the *Data-Handling* AP is executed. If they differ, the mode change is executed. If the next mode needs to adapt the logic in the PL, the `reconfig_required_flag` is set to one. The required reconfiguration requests are then enqueued and when the system switches into a `on_mode_change_mode`. Then, when the *reconfiguration AP* sets the `reconfig_required_flag` to zero, it means that the reconfigurations requested have been solved and the system enters in the requested mode. Once the PL has been adapted (i.e., reconfigured), the mode change is performed by updating the `current_mode` variable.

### B. ARTICo$^3$ PRIMITIVES IN THE MISSION APs

The APs related to a given application must follow the formalized scheme showed in Algorithm 1, which has been extended with the invocation of ARTICo$^3$ functions in Algorithm 2.

The *config* command is sent to all APs once the `artico3_init` and `artico3_kernel_create` ARTICo$^3$ initialization functions are executed during the

**Algorithm 2** ARTICo$^3$ Runtime Calls in Application APs

```
thread AP_<NAME>_THREAD {
rtems_rate_monotonic_create(AP_<name>_period):
loop:
    rtems_rate_monotonic_period(AP_<name>_period)
    function AP_<name>_receive() {
        if new_messages == TRUE then
            if new_message == config_message then
                artico3_alloc("port_A")
                artico3_alloc("port_B")
                artico3_alloc("...")
            end if
            if new_message == close_message then
                close_flag_ = true
            end if
        end if
    end function
    function AP_<name>_step() {
        if close_flag_ == false then
            Initialize data buffers
            artico3_execute("kernel_A")
            Store results
        else
            artico3_free("port_A")
            artico3_free("port_B")
            artico3_free("...")
        end if
    end function
    function AP_<name>_publish() {
        if new_output_data == TRUE then
            publish_data()
        end if
    end function
    goto loop.
}
end thread
```

RTEMS initialization. When an application AP receives this message, it initializes all the ARTICo$^3$ ports used by the kernel associated with that AP. When the message received is the *close* command, a flag is triggered to signal that the AP is not going to be used anymore in the mission, and therefore it can free the memory associated with the ARTICo$^3$ ports when all data transactions to hardware accelerators have already finished.

If the system is working on a mode in which the AP is enabled, it performs the required computations using the hardware accelerators. In this situation, the ARTICo$^3$ input buffers (i.e., the allocated ARTICo$^3$ ports) must be initialized. Data is sent, processed, and received by invoking the `artico3_execute` function. Then, the AP must store the results of the computation. If the workload exceeds the processing capabilities of the hardware accelerators in a single round, several transactions are transparently issued and managed by the ARTICo$^3$ runtime library.

## VII. FAULT-MITIGATION TECHNIQUES

This section describes the FDIR techniques integrated into the on-board processor to ensure the survival of the COTS device along the whole mission time. These include accelerator-level hardware redundancy, device primitives capable of detecting SEUs in the configuration memory, and a custom scrubber specifically designed for reconfigurable systems.

Firstly, the ARTICo$^3$ framework provides inherent fault tolerance in the reconfigurable partitions through accelerator-level redundancy. As explained in section IV-B, the ARTICo$^3$ voter masks and detects faults by comparing the outputs provided by all the replicated accelerators. However, this is not a repair mechanism, so it is not enough for really harsh environments such as outer space, in which too much cumulative faults would eventually make all the accelerators to fail. Moreover, this protection excludes the ARTICo$^3$ static infrastructure and any other application-specific circuitry included in the PL.

In turn, Zynq UltraScale+ devices have dedicated logic primitives to detect and correct SEUs using error redundancy codings. In particular, each configuration frame is protected by an Error Correction Code (ECC) that supports the correction of configuration memory with up to 4-bit errors, whereas a Cyclic Redundancy Check (CRC) assesses the integrity of the whole configuration memory [32]. To adopt these features, the SEM-IP core, which performs all the operations needed to locate errors, has to be instantiated in the design. Beyond locating them, the SEM-IP can directly repair most faults by using the ECC mentioned above. However, in the case of multiple-bit upsets distributed with some particular patterns, the SEM-IP only reports the position of the affected frames, but it is not capable of correcting them. Moreover, some combinations of faults can only be detected by the CRC and not by the ECC, and so even the positions of the damaged frames are unknown.

Finally, a reconfiguration-aware scrubber has been implemented to identify the position of the faults detected but not located by the SEM-IP. This scrubber reads back the content of the configuration memory and compares the read data with a golden copy stored in external memory. A set of mask files (generated during the synthesis of the platform) is used to identify the bitstream positions truly controlling the circuit configuration. Since the system can be adapted at run-time, the golden copy to be used also varies with time. The proposed scrubber uses a table that reflects the modules already loaded in the reconfigurable regions. This table is updated every time an accelerator is reconfigured.

All those fault mitigation techniques are supported from the on-board software perspective by two APs: on the one hand, the *Fault-Detection AP*, which monitors the output of the SEM-IP and the ARTICo$^3$ error monitors; on the other hand, the *Reconfiguration AP*, in which configuration memory is returned to its original state when the SEM-IP reports a non-reparable (or even not located) fault. The *Fault-Detection AP* triggers a mode change to enter in *Repair Mode* when needed. In this mode, the area of the FPGA affected by the error is reconfigured to fix the possible faults. In the case of non-located faults, the *Reconfiguration AP* first executes the reconfiguration-aware scrubber, to know which is the affected region.

The FDIR tasks have been allocated in the real-time processors. The lockstep mode of operation, together with the use of RTEMS, makes it an increased reliability fabric in the Zynq UltraScale+ device. As discussed in [36], the Zynq UltraScale+ also includes a PMU, a fault-tolerant triple-redundant processor, which could also be used as a reliable fabric to run the scrubbers. However, no RTEMS support is available for the PMU, making it incompatible with space scenarios.

## VIII. VBN APPLICATION MAPPING IN THE ON-BOARD PROCESSOR

This section describes how the VBN application has been integrated into the proposed OBP reconfigurable architecture. The proposed VBN implementation exploits the heterogeneous nature of the reconfigurable on-board processor, resulting in a software/hardware solution. The most computationally demanding tasks are implemented as ARTICo$^3$ reconfigurable hardware accelerators in the PL.

Figure 3 shows the floorplanning of the reconfigurable OBP, including the ARTICo$^3$ reconfigurable slots and the SEM-IP. Six reconfigurable slots have been included in the PL, two large ones (named *a3_slot_0* and *a3_slot_3*), and four small ones (named *a3_slot_1*, *a3_slot_2*, *a3_slot_4* and *a3_slot_5*).

Four different accelerators have been implemented for the application. Two of them (*AbsNav_s1* and *AbsNav_s2*) are dedicated to the image processing required by the *absolute navigation* mode. Due to data dependencies, these accelerators are invoked sequentially, one after the other, during each navigation iteration. This approach provides better results in terms of hardware resource utilization, showing the benefits of dynamic and partial reconfiguration technique for in-flight silicon reuse. For this reason, the accelerators are reconfigured at run-time during each navigation period, so both share a single slot. Considering the amount of logic required by these accelerators, they are configured in any of the small slots available in the FPGA. The process begins filling input data memories of *AbsNav_s1*; then, the corresponding kernel execution is done in the FPGA. The results are written in output data memories and collected from it. At this point, an intermediate DPR takes place replacing the *AbsNav_s1* implemented with *AbsNav_s2*. Once the FPGA is ready, the second phase begins, filling *AbsNav_s2* input memories with the results obtained in the first stage. The next steps are execution and final results data collection. To finish the process, a new reconfiguration is needed to implement *AbsNav_s1* in the FPGA to be ready for the next burst. The first part of Absolute Navigation computes the image

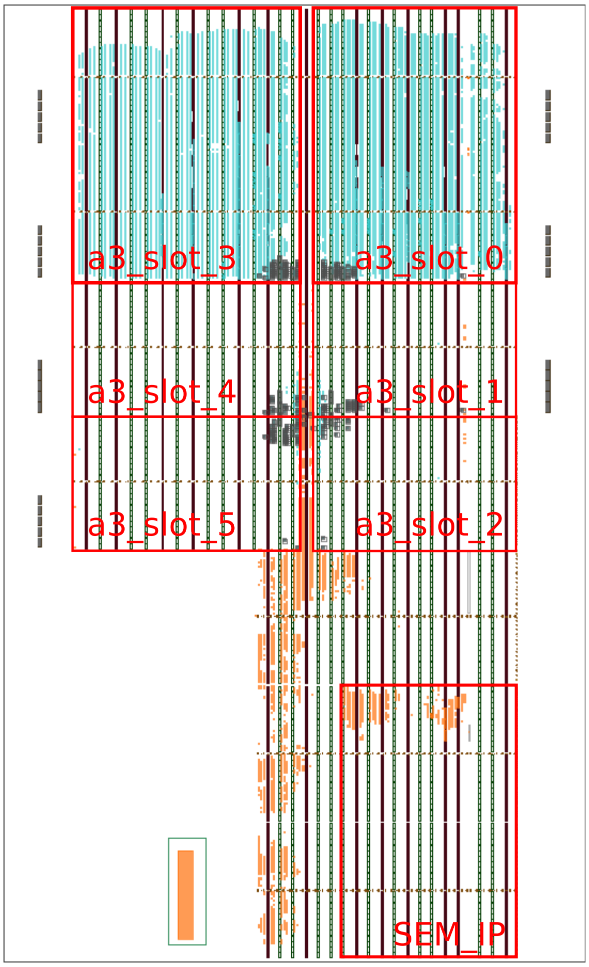

**FIGURE 3. Reconfigurable OBP floorplan.**

filtering convolutions to extract the initial image of edges. The second state iterates over these edges to refine the solution, searching for circular shapes to be recognized as craters.

In turn, the *relative navigation* mode requires of the *Relnav_s1* and the *Relnav_s2* accelerators. Both are executed in parallel to improve performance, so they are configured in two different ARTICo$^3$ slots. They are configured in the large slots of the device. To process one complete 512 × 512 pixels image, 128 cycles need to be carried out. In turn, 512 cycles are needed for 1024 × 1024 pixel images. Each cycle involves filling input memories, FPGA execution, and collecting data from output memories. Finally, the Kalman filter in charge of the fusion of the navigation data produced by the vision algorithms and the measurements produced by external inertial sensors is carried out by the *GNC* AP, which is enabled in all the modes of the application requiring navigation.

Since a large MPSoC has been selected for the implementation of the reconfigurable OBP, the application does not occupy all the slots available in the device. Free slots can be used within the proposed architecture to other applications to be included on-board, such as earth observation or communications.

The time consumed in the reconfiguration is a primary constraint due to the space mission characteristics as the systems are built as hard real-time, so the image processor shall deliver the processes data within a defined period.

## IX. IMPLEMENTATION RESULTS

Experimental results have been carried out to evaluate the proposed reconfigurable on-board processor in the VBN use case. The parameters selected to measure the overhead introduced by the DPR technique with the ARTICo$^3$ architecture are the number of unused resources by using fixed-size slots and the reconfiguration time per accelerator. Moreover, the CPU occupancy required by the FDIR techniques used to protect the COST device is reported.

### A. RESOURCE OCCUPANCY

Table 2 shows the total logic resources available per reconfigurable slot in the platform, together with the sizes of the associated configuration bitstreams. It can be seen that the large slots (*a3_slot_0* and *a3_slot_3*) have approximately double resources than the small ones. As expected, the size of the partial bitstreams is proportional to the number of resources existing in the reconfigurable region.

**TABLE 2.** Resource occupancy per slot.

| Slot | LUTs | FFs | BRAM18s | DSPs | Bitstream Size [MB] |
|---|---|---|---|---|---|
| a3_slot_0 | 108480 | 86400 | 576 | 384 | 4.2 |
| a3_slot_1 | 54240 | 43200 | 288 | 192 | 2.1 |
| a3_slot_2 | 54240 | 43200 | 288 | 192 | 2.1 |
| a3_slot_3 | 109440 | 86400 | 576 | 432 | 3.8 |
| a3_slot_4 | 54720 | 43200 | 288 | 216 | 1.9 |
| a3_slot_5 | 54720 | 43200 | 288 | 216 | 1.9 |

**TABLE 3.** Resource occupancy per kernel.

| IP | LUTs | FFs | BRAM18s | DSPs |
|---|---|---|---|---|
| RelNav | 22707 | 19235 | 87 | 16 |
| RelNav_2 | 2721 | 2975 | 72 | 19 |
| AbsNav_s1 | 1285 | 5557 | 62 | 99 |
| AbsNav_s2 | 4390 | 4633 | 76 | 6 |
| Stereo-Vision | 8063 | 7286 | 66 | 0 |

In turn, table 3 shows the logic resources occupied by each kernel in the VBN application. It is worth noting the high number of resources required to implement the *relnav_detection* algorithm compare to the others. The large slots in the platform have been defined considering the requirements of this accelerator, which produces underuse of resources when these slots allocate the other accelerators.

The relative usage of resources, when mapping the algorithms into ARTICo$^3$ slots, is shown in table 4. The results have been grouped per accelerator and slot type (large or small). As a simplification, only the (*a3_slot_0* and *a3_slot_1*) have been used to obtain the presented values for the large and small slots, respectively. Given these results, it is necessary to mention that the limitation preventing the allocation of the *relnav_s1* accelerator in the small slots does not come from a lack of resources, but from the impossibility shown by the vendor CAD tools to route the accelerator netlist under a high occupancy in the slot.

**TABLE 4.** Relative resource occupancy per kernel.

| kernel | slot | % LUTs | % FFs | % BRAM18s | % DSPs |
|---|---|---|---|---|---|
| RelNav | large | 20.93 | 22.26 | 15.10 | 4.17 |
| | small | 41.86 | 44.53 | 30.21 | 8.33 |
| RelNav_ 2 | large | 2.51 | 3.44 | 12.50 | 4.95 |
| | small | 5.02 | 6.30 | 25.00 | 9.90 |
| AbsNav_s1 | large | 1.18 | 6.43 | 10.76 | 25.78 |
| | small | 2.37 | 12.86 | 21.53 | 51.56 |
| AbsNav_s2 | large | 4.05 | 5.36 | 13.19 | 1.56 |
| | small | 8.09 | 10.72 | 26.39 | 3.13 |
| Stereo-Vision | large | 7.43 | 8.43 | 11.46 | 0 |
| | small | 14.87 | 16.87 | 22.92 | 0 |

### B. RECONFIGURATION TIME

The time required to load an accelerator in the ARTICo$^3$ architecture only depends on the size of the slot, regardless it is fully occupied or not. Therefore, table 5 shows the time required for the reconfiguration of each slot in the proposed platform. Apart from the time needed for downloading the bitstream in the device configuration memory (shown in the *DPR* column in the table), whenever a new accelerator is downloaded in ARTICo$^3$, the table maintained by the reconfiguration-aware scrubber must be updated. The time needed to update this table is shown in the *Update GCs*. The total time required for updating an accelerator is, therefore, the sum of the *DPR* and *Update GCs* columns. All these tests have been carried out with the PCAP working at 187.5MHz, the maximum achievable frequency with the default PLL assigned to this interface.

**TABLE 5.** Reconfiguration times.

| Slot | Update GCs [ms] | DPR [ms] | Total Time [ms] |
|---|---|---|---|
| a3_slot_0 | 18.59 | 9.26 | 27.85 |
| a3_slot_1 | 1.60 | 4.64 | 6.24 |
| a3_slot_2 | 1.62 | 4.65 | 6.27 |
| a3_slot_3 | 23.65 | 8.52 | 32.17 |
| a3_slot_4 | 3.29 | 4.26 | 7.55 |
| a3_slot_5 | 1.73 | 4.26 | 5.99 |

### C. VBN NAVIGATION PERFORMANCE

The corresponding execution time parameters of the mission-dependent APs can be seen in table 6, which have been measured exhaustively in the application. The hardware accelerators associated to *RelNav*, *RelNav_2*, *AbsNav* and *Stereo-Vision* APs have being simplified for the frame of this work in order to fit in the designed ARTICo$^3$ slots and the

**TABLE 6.** Mission-dependent APs execution times.

| AP | mean time [s] | BCET [s] | WCET [s] |
|---|---|---|---|
| Relnav | 0.336 | 0.264 | 0.363 |
| Relnav 2 | 0.402 | 0.07 | 0.486 |
| Absnav | 5.2 | 4.6 | 5.9 |
| Stereo-Vision | 0.325 | 0.293 | 0.374 |

overall MPSoC architecture, that includes data transmission from memory between the accelerators and software tasks.

### D. FAULT DETECTION AND REPAIR CAPABILITIES

The use of COTS devices, beyond the clear benefit offered in terms of performance, adaptability and cost, has an impact in the form of CPU occupancy due to the FDIR techniques required to guarantee the survivability of the system. This overhead is measured and analyzed in this subsection.

In the case of the proposed platform, the fault detection time heavily depends on the time required by the SEM-IP to detect a fault. The value provided by the Xilinx Product Guide [32] to detect a fault by ECC is 28ms when using the configuration memory interface employed by the SEM-IP (ICAP) at its maximum operating frequency, which corresponds to 200MHz. Moreover, the worst-case corresponds to the CRC non-locatable faults, which is twice the time to detect a fault using ECC, so it is 112ms ($WC_{SEMDet}$) in the present solution, where the ICAP works at 100 MHz. However, the proposed system will not react until the *Fault-Detection AP* checks the output of this peripheral. As the errors can appear in the configuration memory just after the execution of the *Fault-Detection AP*, the system would not detect the faults until the next execution of this task. Hence, the WCET of the *Fault-Detection AP* ($WC_{FDAP}$) and its execution period ($T_{FDAP}$) must be taken into account, which is expressed in the following equation:

$$WC_{Det} = WC_{SEMDet} + WC_{FDAP} + T_{FDAP} \tag{1}$$

where $WC_{Det}$ refers to the worst case for the fault detection time. Hence, the time needed to react to a fault occurrence, considering the hardware and software systems involved.

**TABLE 7.** Repair times.

| Err type | mean time [ms] | BCET [ms] | WCET [ms] |
|---|---|---|---|
| Repair per frame | 0.228 | 0.048 | 0.355 |
| Readback | 1778.87 | 1775.74 | 1785.24 |

Regarding fault correction, the time required to repair the device actively is shown in table 7. These metrics correspond to the repair actions that the *Reconfiguration AP* can perform when multiple-bit upsets, non-reparable by the SEM-IP, are detected in the configuration memory. On the one hand, if the faults are located but not repaired by the SEM-IP, the *Reconfiguration AP* corrects the affected frame, which takes 0.355 ms in the worst case, as shown in table 7. On the other hand, if the SEM-IP cannot even locate the fault, it is necessary to trigger the readback scrubbing of the whole configuration memory, as explained in section VII. This operation takes 1785.24 ms, in the worst case ($WC_{RCAP}$).

Likewise to the fault detection, in the case of fault correction, the period of the *Reconfiguration AP* ($T_{RCAP}$) together with its WCET ($WC_{RCAP}$) must be considered to obtain the worst case for the correction time ($WC_{Corr}$). This metric can be obtained with the following expression:

$$WC_{Corr} = WC_{RCAP} + T_{RCAP} \tag{2}$$

The maximum periods that can be assigned to the FDIR tasks ($T_{RCAP}$ and $T_{FDAP}$) depend on the satellite orbit where it will operate, which will impact on the expected failure rate, combined with the device sensitivity metric to single event effects. The shortest are these periods, the lower will be the CPU time available for the application-dependent threads. Considering the assumptions of the classical rate monotonic scheduling algorithm [35], where it is formulated that the execution time of every thread must be considered constant, the CPU utilization percentage for the FDIR tasks, can be determined using the periods assigned to both the *Fault-Detection* and *Reconfiguration APs*, together with their WCET, as expressed in the following equation:

$$U_{FDIR} = \frac{WC_{FDAP}}{T_{FDAP}} + \frac{WC_{RCAP}}{T_{RCAP}} \tag{3}$$

With these FDIR AP parameters, the overhead introduced by the FDIR tasks can be computed in terms of CPU utilization. Unless redundancy is applied to the accelerators, the occurrence of SEUs during the computation of an accelerator will produce discarding its execution results. In particular, the worst-case conditions will happen when:

- Faults are detected just before the end of an execution of the VBN task, so the whole computation time ($WC_{VBN}$) of the task will be useless.
- Faults occur just after the execution of the FDIR tasks, so the whole period assigned to the FDIR task ($T\_FDIR$) has to be waited.

On top of that, in the event of a fault, a correct computation (with no errors) of the VBN algorithms would be delayed the time spent in detecting and fixing the fault ($WC_{Det} + WC_{Corr}$) in addition to the period assigned to the VBN task ($T_{VBN}$). Considering these parameters and the occurrence of a fault under the worst-case conditions, the time expected to perform an error-free execution of the accelerators ($t_{EFE}$) can be obtained with the following relationship:

$$t_{EFE} > 2 * WC_{VBN} + T_{VBN} + WC_{Det} + WC_{Corr} \tag{4}$$

### E. COMPARISON WITH PREVIOUS WORKS IN THE STATE-OF-THE-ART

This subsection provides a comparison of the most relevant features of the on-board processor architecture proposed in this paper, with significant works in the state-of-the-art. The platforms provided in [13] and [15] have been selected for

**TABLE 8. Platform comparison.**

| Platform | MPSoC | Processors | OS | DPR usage | HW Logic redundancy | Scrubbers |
|---|---|---|---|---|---|---|
| CSP [13] | Zynq-7020 | ARM-A9 | Linux<br>RTEMS compatible | Adaptation<br>Reparation | Redundancy framework compatible | Hybrid Scrubber Compatible |
| APEX-SoC [15] | Zynq-7100 | ARM-A9 | Linux | Reparation | DMR on processing pipelines<br>Low-level redundancy on critical parts | Blind Full FPGA<br>ECC-CRC |
| Proposed platform | ZU9EG | ARM-R5-lockstep | RTEMS | Adaptation<br>Reparation | Dynamic, at accelerator level | Readback Cfg Aware<br>ECC-CRC |

this comparison, since they also include mission and fault mitigation techniques, all integrated. These three platforms have been developed as baseline architectures to be reused in different missions, and they are all based on MPSoCs. Other works have been discarded because they do not present a similar level of integration or they have used other types of devices (i.e., radiation-hardened FPGAs). Table 8 gathers the main features of the selected works and the one presented in this paper.

Differently to the reference platforms, the proposal described in this paper is implemented on a Zynq Ultrascale+ device in which the R5 processors are used as the primary computing engine, working in lock-step mode. Regarding the OS support, authors in [13] claim their platform is RTEMS-compatible, but this is not detailed in any specific implementation. Differently, in our proposal, we have fully described the porting of RTEMS to the Zynq Ultrascale+ device. Moreover, we have included dynamically scalable hardware accelerators, which also provide selectable fault tolerance levels. Generic templates are also provided to ease the deployment of mission-specific applications.

The proposed platform is also featured with a novel readback scrubber that is compatible with DPR. The proposed scrubber does not require the reconstruction of the golden copy, as happens in [14] or [23]. Instead, the scrubber selectively accesses the full and different partial golden copies when it needs to validate areas of the configuration memory that have been modified using more than one bitstream. Regarding the performance of the proposed reconfiguration-aware readback scrubber, the results presented in [22] can be used as a reference. They have experimentally measured the time required for detecting and correcting different types of faults. With around 6.5 times smaller configuration memory, the time required to detect bit upsets using ECC codes is 7 times smaller than in our work, so performance is roughly equivalent. Regarding correction, the time needed to correct frames affected by multiple bit upsets non-reparable by the SEM-IP is lower in our proposal, even when the repair mechanism is aware of the current device configuration. This improved performance includes the overhead in the recovery mechanism of looking up the required bitstream to fetch the frame to correct. In the case of CRC error, the full FPGA memory is verified, which requires approximately two seconds in both approaches. However, it must be noticed that equivalent performance is better since the configuration memory is 6.5 times bigger in our proposal.

## X. CONCLUSIONS AND FUTURE WORK

This paper describes a novel architecture for a reconfigurable on-board processor for space applications that aims to exploit the flexibility and cost benefits provided by COTS SRAM-based MPSoCs. Dynamic reconfiguration is used to provide in-flight adaptation and as a fault repair mechanism. Reconfiguration is offered by the integrated ARTICo$^3$ infrastructure, which provides transparent adaptation of mission-specific hardware accelerators, as well as, scalable performance and inherent fault tolerance in the accelerators. A reconfiguration-aware scrubber has also been included as a global FDIR technique, that can ultimately guarantee the survival of the platform during the whole mission time. Besides the hardware setup, the RTEMS real-time operating system has been integrated to schedule and coordinate the deterministic execution of all the software tasks. A portable template is also provided for each task included in the system, which, together with the ARTICo$^3$ toolchain, makes it easier to integrate and manage mission-dependent and support tasks. The platform benefits are shown for a visual-based navigation use-case, in which the platform reconfiguration is used to adapt the navigation algorithms at run-time, depending on the operational stage the spacecraft is. Experimental results show how real-time support allows adjusting the availability of the platform to the requirements of the application, having the mission orbit, and the device manufacture technology as design parameters. The implications of using the DPR technique have also been quantified, in terms of internal fragmentation, reconfiguration time, and repair times.

The future work will focus on providing a theoretical study on the availability of the logic inside each particular accelerator, as well in using the device reconfiguration function to carry out exhaustive fault-injection campaigns to validate the models in the practice. More in-depth research will be carried out on real-time scheduling algorithms to increase the availability of the platform. In addition, new applications in the space domain will be integrated in the platform, including earth observation and communications, among others.

## REFERENCES

[1] E. S. Agency. *Rosetta Mission*. Accessed: Nov. 20, 2019. [Online]. Available: https://www.esa.int/Our_Activities/Space_Science/Rosetta

[2] NASA. *Mars Insight Mission*. Accessed: Nov. 20, 2019. [Online]. Available: https://mars.nasa.gov/insight/timeline/landing/entry-descent-landing/

[3] S. Y. Chen, "Kalman filter for robot vision: A survey," *IEEE Trans. Ind. Electron.*, vol. 59, no. 11, pp. 4409–4420, Nov. 2012.

[4] T. M. Lovelly and A. D. George, "Comparative analysis of present and future space-grade processors with device metrics," *J. Aerosp. Inf. Syst.*, vol. 14, no. 3, pp. 184–197, Mar. 2017.

[5] 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," *J. Aerosp. Inf. Syst.*, vol. 15, no. 4, pp. 178–192, Apr. 2018.

[6] G. Martin, "Newspace: The emerging commercial space industry," NASA Ames Res. Center, Moffett Field, CA, USA, Tech. Rep. ARC-E-DAA-TN20859, 2015.

[7] A. Rodríguez, J. Valverde, J. Portilla, A. Otero, T. Riesgo, and E. de la Torre, "FPGA-based high-performance embedded systems for adaptive edge computing in cyber-physical systems: The ARTICo$^3$ framework," *Sensors*, vol. 18, no. 6, p. 1877, 2018.

[8] H. Quinn, E. Johnson, J. Johnson, B. Pratt, N. Rollins, J. Krone, D. Roussel-Dupre, M. Caffrey, P. Graham, M. Wirthlin, K. Morgan, A. Salazar, T. Nelson, and W. Howes, "The cibola flight experiment," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 8, no. 1, pp. 1–22, Mar. 2015, doi: 10.1145/2629556.

[9] D. Ratter, "FPGAs on mars," *Xcell J.*, vol. 50, pp. 8–11, Aug. 2004.

[10] F. Rittner, R. Glein, T. Kolb, and B. Bernard, "Broadband FPGA payload processing in a harsh radiation environment," in *Proc. NASA/ESA Conf. Adapt. Hardw. Syst. (AHS)*, Jul. 2014, pp. 151–158.

[11] *Correct the Word Processo With Processor*. Accessed: Nov. 28, 2019. [Online]. Available: https://www.iis.fraunhofer.de/en/ff/kom/satkom/FOBP.html

[12] A. Hofmann, R. Glein, L. Frank, R. Wansch, and A. Heuberger, "Reconfigurable on-board processing for flexible satellite communication systems using FPGAs," in *Proc. Topical Workshop Internet Space (TWIOS)*, Jan. 2017, pp. 1–4.

[13] C. Wilson and A. George, "CSP hybrid space computing," *J. Aerosp. Inf. Syst.*, vol. 15, no. 4, pp. 215–227, Apr. 2018.

[14] D. Rudolph, C. Wilson, J. Stewart, P. Gauvin, A. George, H. Lam, G. Crum, M. Wirthlin, A. Wilson, and A. Stoddard, "CSP: A multifaceted hybrid architecture for space computing," NASA Goddard Space Flight Center, Greenbelt, MD, USA, Tech. Rep. GSFC-E-DAA-TN15946, 2014.

[15] X. Iturbe, D. Keymeulen, E. Ozer, P. Yiu, D. Berisford, K. Hand, and R. Carlson, "An integrated SoC for science data processing in next-generation space flight instruments avionics," in *Proc. IFIP/IEEE Int. Conf. Very Large Scale Integr. (VLSI-SoC)*, Oct. 2015, pp. 134–141.

[16] eoPortal Directory. *Ops-Sat (Operations Nanosatellite)*. Accessed: Dec. 5, 2019. [Online]. Available: https://directory.eoportal.org/web/eoportal/satellite-missions/o/ops-sat

[17] F. Bezerra, D. Dangla, F. Manni, J. Mekki, D. Standarovski, R. G. Alia, M. Brugger, and S. Danzeca, "Evaluation of an alternative low cost approach for SEE assessment of a SoC," in *Proc. 17th Eur. Conf. Radiat. Its Effects Compon. Syst. (RADECS)*, Oct. 2017, pp. 1–5.

[18] eoPortal Directory. *Eyesat 3u Cubesat Astronomy Mission to Study Zodiacal Light*. Accessed: Dec. 5, 2019. [Online]. Available: https://directory.eoportal.org/web/eoportal/satellite-missions/o/ops-sat

[19] M. Meftah, E. Bamas, P. Cambournac, P. Cherabier, R. Demarets, G. Denis, A. Dion, R. Duroselle, F. Duveiller, and L. Eichner, "SERB, a nano-satellite dedicated to the Earth-sun relationship," *Proc. SPIE Sensors Syst. Space Appl. IX*, vol. 9838, May 2016, Art. no. 98380T.

[20] N. Montealegre, D. Merodio, A. Fernandez, and P. Armbruster, "In-flight reconfigurable FPGA-based space systems," in *Proc. NASA/ESA Conf. Adapt. Hardw. Syst. (AHS)*, Jun. 2015, pp. 1–8.

[21] eoPortal Directory. *Fedsat (Federation Satellite)*. Accessed: Dec. 5, 2019. [Online]. Available: https://directory.eoportal.org/web/eoportal/satellite-missions/o/ops-sat

[22] A. Stoddard, A. Gruwell, P. Zabriskie, and M. J. Wirthlin, "A hybrid approach to FPGA configuration scrubbing," *IEEE Trans. Nucl. Sci.*, vol. 64, no. 1, pp. 497–503, Jan. 2017.

[23] J. Heiner, B. Sellers, M. Wirthlin, and J. Kalb, "FPGA partial reconfiguration via configuration scrubbing," in *Proc. Int. Conf. Field Program. Log. Appl.*, Aug. 2009, pp. 99–104.

[24] K. Maragos, V. Leon, G. Lentaris, D. Soudris, D. Gonzalez-Arjona, R. Domingo, A. Pastor, D. M. Codinachs, and I. Conway, "Evaluation methodology and reconfiguration tests on the new European NG-MEDIUM FPGA," in *Proc. NASA/ESA Conf. Adapt. Hardw. Syst. (AHS)*, Aug. 2018, pp. 127–134.

[25] K. Bravhar, V. Martins, L. Santos, and D. M. Codinachs, "BRAVE NG-MEDIUM FPGA reconfiguration through space wire: Example use case and performance analysis," in *Proc. NASA/ESA Conf. Adapt. Hardw. Syst. (AHS)*, Aug. 2018, pp. 135–141.

[26] D. M. Hiemstra, V. Kirischian, and J. Brelski, "Single event upset characterization of the zynq UltraScale+ MPSoC using proton irradiation," in *Proc. IEEE Radiat. Effects Data Workshop (REDW)*, Jul. 2017, pp. 1–4.

[27] T. L. I. Technologies. *Single Event Characterization of a Xilinx Ultrascale+ MP-SOC FPGA*. Accessed: Dec. 9, 2019. [Online]. Available: https://indico.esa.int/event/232/contributions/2162/

[28] X. Iturbe, B. Venu, E. Ozer, and S. Das, "A triple core lock-step (TCLS) ARM cortex-R5 processor for safety-critical and ultra-reliable applications," in *Proc. 46th Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw. Workshop (DSN-W)*, Jun. 2016, pp. 246–249.

[29] Microchip. *Rad-Tolerant/Rad-Hard Integrated Circuits*. Accessed: Jan. 14, 2020. [Online]. Available: http://ww1.microchip.com/downloads/en/DeviceDoc/00002200C.pdf

[30] Xilinx. *Ultrascale Architecture Configuration—User Guide*. Accessed: Dec. 16, 2019. [Online]. Available: https://www.xilinx.com/support/documentation/user_guides/ug570-ultrascale-configuration.pdf

[31] H. Hirschmuller, "Accurate and efficient stereo processing by semi-global matching and mutual information," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2005, pp. 807–814.

[32] Xilinx. *Ultrascale Architecture Soft Error Mitigation Controller V3.1—Product Guide*. Accessed: Dec. 19, 2020. [Online]. Available: https://www.xilinx.com/support/documentation/ip_documentation/sem_ultra/v3_1/pg187-ultrascale-sem.pdf

[33] R. Ginosar, "Survey of processors for space," in *Proc. Data Syst. Aerosp. (DASIA)*, 2012, pp. 1–5.

[34] Xilinx. *Xilinx Sem-Ip*. Accessed: Jan. 20, 2020. [Online]. Available: https://www.xilinx.com/products/intellectual-property/sem.html

[35] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *J. ACM*, vol. 20, no. 1, pp. 46–61, 1973.

[36] A. Perez, A. Otero, and E. de la Torre, "Performance analysis of SEE mitigation techniques on zynq ultrascale+hardened processing fabrics," in *Proc. NASA/ESA Conf. Adapt. Hardw. Syst. (AHS)*, Aug. 2018, pp. 51–58.

**ARTURO PÉREZ** received the B.Sc. degree in industrial electronics and automation engineering from the University Carlos III of Madrid (UC3M), Spain, in 2015, and the M.Sc. degree in industrial electronics from the Universidad Politécnica de Madrid (UPM), in 2017, where he is currently pursuing the Ph.D. degree. He joined CEI, in 2016, as an Associated Researcher to the ENABLES3 European Project. His current research interests include reconfigurable computing, embedded systems, real-time systems, and singleevent-effects mitigation.

**ALFONSO RODRÍGUEZ** (Student Member, IEEE) received the B.Sc. degree in industrial engineering and the M.Sc. degree in industrial electronics from the Universidad Politécnica de Madrid (UPM), Madrid, Spain, in 2012 and 2014, respectively, where he is currently pursuing the Ph.D. degree in industrial electronics with the Centro de Electrónica Industrial. Under a HiPEAC collaboration grant, he was a Visiting Researcher with the Computer Engineering Group, Universität Paderborn, where he worked on operating systems for reconfigurable computing. He is also a Teaching Assistant and a full-time Researcher of industrial electronics with the Centro de Electrónica Industrial, UPM. His current research interests include artificial intelligence, high-performance embedded systems, and reconfigurable computing.

**ANDRÉS OTERO** (Member, IEEE) received the M.Sc. degree (Hons.) in telecommunication engineering from the University of Vigo, in 2007, and the master's and Ph.D. degrees in industrial electronics from the Universidad Politécnica de Madrid (UPM), in 2009 and 2014, respectively. He is currently an Assistant Professor of electronics with UPM, and a Researcher with the Centro de Electrónica Industrial (CEI). His current research interests include embedded system design, reconfigurable systems on FPGAs, evolvable hardware, and embedded machine learning. During the last years, he has been involved in different research projects in these areas, and he is the author of more than 30 articles published in international conferences and journals. He has served as a Program Committee Member of different international conferences in the field of reconfigurable systems, such as ARC, ReConFig, DASIP, and ReCoSoC.

**DAVID GONZÁLEZ ARJONA** received the M.Phil. degree in telecommunications engineering and computer science and the M.S. degree in computer science from the Autonoma University of Madrid. He works with GMV Aerospace and Defence SAU while in parallel works part-time as an Associate Professor with the Autonoma University of Madrid. He has been specialized in Avionics and On-Board SW for the Space Segments and Robotics Business Unit of GMV, since 2011. He is currently the Head of the Microelectronics Section, leading as directly as the Manager different ESA projects related to Autonomous Vision-Based Navigation, Communication, Avionics Computers and participating in R&D projects as ENABLE-S3 H2020 or Space-Based Surveillance Tracking on Galileo Next Generation. He is also a part-time Associate Professor teaches with the Department of Electronic and Telecommunication Technology providing a state-of-the-art point of view from academic and researching perspective.

**ÁLVARO JIMÉNEZ-PERALO** received the degree in computers engineering from the Autonoma University of Madrid. He has been a Project Manager at GMV Aerospace and Defence SAU on the Avionics & On-Board SW for the Space Segments and Robotics Business Unit of GMV, since 2017. He is working in projects involving development of embedded SW for different on board applications as Exomars 2020, including UML design, unit testing and documentation as well as projects with embedded SW and FPGA as LVDAC, where a development over a Zynq board imply an AOCS running on processor and interacting with FPGA IPs. He is leading as the Manager of the GMV participation in the project Multispectral Navigation Camera with applicability to HERACLES LOP-G rendezvous and docking. He is also working at the research projects for European H2020 as ENABLE-S3, where a fast validation and verification using a reconfigurable HW architecture is used to manage different redundancy of HW accelerators.

**MIGUEL ÁNGEL VERDUGO** received the M.Sc. degree in industrial engineering from the Technical University of Madrid (UPM), Spain. He is currently an Engineer with the Space Segment and Robotics Department assigned to different projects related to microelectronics firmware and embedded SW under ESA contracts. He has been involved in several FPGA designs of firmware IP-cores for on-board computers modules as for the in-flight partial dynamic reconfiguration of ENABLE-S3 project. He has experience working in the space sector, especially with FPGA technology, and a deep experience in VHDL. He was involved in Mars Sample Return Project devoted to the HW-acceleration of a shortrange image processing function for model-match of the OS canister capture.

**EDUARDO DE LA TORRE** received the Ph.D. degree in electrical engineering from the Universidad Politécnica de Madrid (UPM), Spain, in 2000. He is currently an Associate Professor of electronics with the Centre of Industrial Electronics, UPM. His main interests include FPGA design, embedded systems design, HW acceleration, signal processing and partial, and dynamic reconfiguration of digital systems. He has participated in more than 40 projects, 11 of them being EU funded projects, and overall, in nine funded projects related with reconfigurable systems. He has been a Program Chair of ReConFig and a General Chair of ReCoSoC, two conferences with strong interest in hardware reconfiguration.