SCALABILITY OF A BASE LEVEL DESIGN FOR AN ON-BOARD-COMPUTER FOR SCIENTIFIC MISSIONS

Carl Johann Treudler, Jan-Carsten Schröder, Fabian Greif, Kai Stohlmann, Gökçe Aydos, and Görschwin Fey

Department of Avionics Systems, Institute of Space Systems
German Aerospace Center (DLR), 28359 Bremen, Germany

Email: Carl.Treudler@dlr.de, Jan-Carsten.Schroeder@dlr.de, Fabian.Greif@dlr.de, Kai.Stohlmann@dlr.de, Gockce.Aydos@dlr.de, Goerschwin.Fey@dlr.de

ABSTRACT

Facing a wide range of mission requirements and the integration of diverse payloads requires extreme flexibility in the on-board-computing infrastructure for scientific missions. We show that scalability is principally difficult. We address this issue by proposing a base level design and show how the adoption to different needs is achieved. Inter-dependencies between scaling different aspects and their impact on different levels in the design are discussed.

1. INTRODUCTION

The *On-Board Computer* (OBC) of a space system is typically tailored to a specific mission. In particular, scientific missions exhibit extreme variations in requirements, ranging from small satellites for LEO [1], all the way to extraterrestrial missions [2, 3]. These requirements address functional specifications, interfaces, weight, volume, and energy, as well as many more issues. In particular the eco-system of available subsystems and payloads is non-standardized and extremely volatile for scientific missions. In the past, on-board computers for such missions have often been tailored to an extent that required almost a complete redesign. This has a direct impact on the length of the development cycle and the cost for these missions.

More recently, efforts have been made to promote standardization to ensure reusability of components of the avionics infrastructure. A European example for large space systems is the *Space Avionics Open Interface aRchitecture* (SAVOIR) initiative. SAVOIR takes hardware and software aspects into account to define an avionics reference architecture [4]. However, this standard is mainly a specification guideline, subsuming a wide range of potential designs without the focus on a specific implementation.

Space Plug-and-play Avionics (SPA) is an alternative concept [5], targeting small satellites and fast integration. The underlying idea is to use standardized interfaces to connect the subsystems. Adding noncompliant subsystems requires to connect these through an interface adapter. Intelligent subsystems as well as required interface adapters induce significant overhead on the avionics system in the context of small satellites.

In this paper, we analyze the scalability of an OBC design. First, the design space and problems with respect to scalability are discussed. Next, we introduce a base-level design and discuss how scalability can be achieved with respect to different aspects. We discuss two possible implementations of the base level design facilitating flight qualification and rapid prototyping, respectively. The base level design is then applied to illustrate three different application scenarios as an instrument controller, a central OBC, or an avionics system based on SpaceWire[6] and *Remote Terminal Units* (RTUs).

2. DIMENSIONS OF SCALABILITY

This section first defines an abstract model for finding an optimal OBC under given requirements. We will then discuss which dimensions are relevant to the resulting multi-objective optimization problem. Finally, we will argue how this justifies the practice of incremental design starting from a base-level architecture.

2.1. Abstract Model

On an abstract level we consider the selection of the "best" OBC as a multi-objective optimization problem. This is similar to the view taken in concurrent engineering [7]. The multi-dimensional search space is configured by all parameters and implementation options relevant to the realization of the system. Each parameter or option is considered as one dimension. Thus, each point in this search space either represents a *realizable OBC* or

Proc. 'DASIA 2014', Data Systems In Aerospace
Warsaw, Poland, 3–5 June 2014 (ESA SP-725, September 2014)

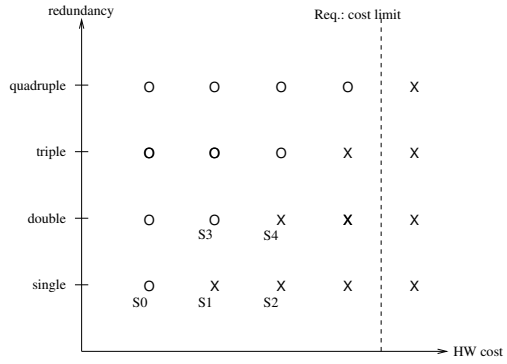


Figure 1. Simple search space

an *unrealizable OBC*. Requirements defined by the mission are broken down into components where each component relates to a single dimension. As a consequence, a requirement defines a hyperplane in the search space that separates those OBCs which are consistent with the requirements from those OBCs which are not consistent with the requirements.

The first question to be answered before implementing the mission is whether there is a realizable OBC in the subspace defined by the requirements – this is a *permissible OBC*. Second, in most cases an optimization problem is solved to find, e.g., the cheapest permissible OBC. The main difficulty in this design task is that the parameters of an OBC cannot be scaled independently.

Example 1 *Consider the simple search space shown in Figure 1 with the two dimensions 'redundancy' and 'hardware cost'. Realizable OBCs are denoted by 'X'; unrealizable OBCs are denoted by 'O'. Obviously, the two dimensions depend on each other. The system without redundancy includes less than half of the hardware compared to the double modular redundant system. Consequently, hardware cost for the nonredundant OBC is lower, roughly half of the cost. In the figure the OBC labeled $S1$ is the cheapest realizable nonredundant OBC. The nonredundant OBC $S0$ at lower cost is not realizable. Building a more expensive non-redundant OBC $S2$ at a higher cost will typically be possible. The cost variation would be due to the exchange of components of the OBC, which is not represented in this simple two-dimensional model. However, building a redundant OBC $S3$ at the same cost as the cheapest non-redundant OBC is not possible, i.e., $S3$ is unrealizable. When increasing the cost, the double modular redundant OBC $S4$ is realizable.*

One requirement puts an upper limit for the cost of the OBC denoted as a dashed vertical line. Each realizable OBC left of this line is a permissible OBC. Thus, the cheapest triple modular redundant OBC is still permissible, while the quadruple modular redundant OBC is not permissible.

This example illustrates how unrealizability, realizability and permissibility are handled in the abstract model. In the simple example, the relation between hardware cost and redundancy can even be modeled relatively accurately. Once the hardware cost for the nonredundant OBC are known, the n-times redundant version will require n times the cost plus an overhead for connecting the redundant parts properly. In other cases, relations may not be as straightforward. This largely depends on the parameters and design options to be handled.

2.2. Parameters and Options

Even the enumeration of all parameters and design options which have to be considered as dimensions in the search space is difficult. The full search space if defined by system level parameters, e.g., reliability, energy consumption, architectural decisions, e.g., interfaces and redundancy concept, functional aspects, e.g., computing performance, memory sizes, and component parameters, e.g., totals dose. Furthermore, development aspects, e.g., engineering cost, hardware cost, risk must be considered.

These dimensions are highly interdependent. Example 1 already illustrated this for the simple two-dimensional case of redundancy versus hardware cost. However, other parameters are much harder to relate to each other. Consider reliability in *Failures in Time* (FIT), where 1 FIT corresponds to one failure in 10^9 hours of operating time. To assess the system level FIT rate where a failure is defined by erroneous output of the system, component level FIT rates must be known, then the full architecture has to be taken into account. This includes hardware and software analysis on how component level failures affect the full system. This is difficult in practice and typically handled by making assumption on the types of faults and the fault propagation within the system [8].

As a consequence, we propose a base level design for an OBC in the following. Considering two mission scenarios, we explain how this base level design may be scaled in several aspects and which parts can be reused.

3. BASE LEVEL DESIGN

Our base level design shown in Figure 2 consists of a CPU, memory, peripherals, multiple communication links and programmable logic. This unit is capable of executing software, storing the software and data, and communicating with outside systems. The programmable logic is connected to the CPU and allows additional peripherals to be implemented. The periphery is presented to the application software through a *Hardware Abstraction Layer* (HAL), which hides most details about the type and configuration of the hardware. Internal power conditioning, analog monitoring, and a watchdog allow for autonomous operation and, consequently, fault isolation properties in many configurations.

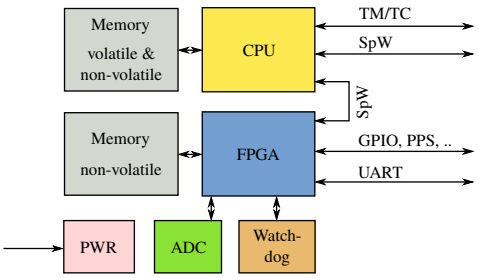


Figure 2. Base Level Design

Table 1. Properties of the implementations

	GR712RC with ProASIC3	FPGA Boards
Cost	high	low
Reliability	high	not quantified
Functionality	flight model equiv.	emulation

The base level design provides a functional unit that can be used for further composition at the architectural level. The instantiation of individual components depends on the application requirements and directly impacts dimensions like cost or reliability. The form-factor is also undefined at this stage as it depends on accommodation requirements and choice of components.

In the following, we will show two implementations of this design. Both implementations provide mostly the same functionality, but for vastly different environmental conditions and are designed for different purposes, but still share the same properties of the base level design. These two implementations are illustrated because they show the flexibility of the design and bridge the gap between COTS and space-qualified systems. Table 1 gives a coarse overview of some properties of the implementations.

3.1. Implementation: GR712RC + ProASIC3

This implementation is intended to be used inside a spacecraft. High-reliability requirements and the radiation environment are addressed by selecting appropriate *Electronic, Electrical and Electromechanical* (EEE) parts as well as local mitigation techniques.

This implementation takes the base level design, shown in Figure 2, and implements it with the following components:

- A GR712RC as CPU
- SDRAM is used as volatile memory
- Non-volatile memory connected to CPU is MRAM
- Non-volatile memory connected to the FPGA is a bank of NAND-Flash devices
- A single flash-based ProASIC3 FPGA as programmable logic

The GR712RC[9] is a *System-on-Chip* (SoC) that is specially designed and built for space applications and their radiation environment. This SoC offers a wide variety of interfaces, ranging from general purpose IOs and UARTs to CCSDS *Telemetry/Telecommand* (TM/TC) interfaces and multiple high-speed SpaceWire ports. The chip also provides decoder/encoder for *Error Correcting Code* (ECC) on the SDRAM and the non-volatile MRAM. Two Leon3FT CPUs provide sufficient computing performance for a wide variety of applications.

Using different memories is a consequence of the requirement for high capacity in volatile and non-volatile form.

A single Microsemi ProASIC3[10] flash-based FPGA is used as programmable-logic. In contrast to anti-fuse based FPGAs, a flash-based FPGA allows the unit to be reconfigured after assembly. Peripheral interfaces are typically implemented as *Intellectual Property* (IP)-cores on the FPGA. Inside the FPGA, IP-cores are connected to an internal bus which in turn is accessed over SpaceWire from the software. In the smallest possible configuration this FPGA only hosts the IP-core controlling the NAND-flash.

The design decision about the interface between the SoC and the FPGA was between SpaceWire and a direct connection to the external memory interface of the GR712RC. SpaceWire was chosen being the more generic solution that allows for easier expansion. This choice also converts the FPGA effectively into an embedded *Remote Terminal Unit* (RTU) and eliminates the dependency to the more specialized external memory interface of the GR712RC.

3.2. Implementation: Two Commercial FPGA Boards

This implementation is based on two commercial and readily available FPGA-development boards. This implementation is used for rapid prototyping intended for early software and IP-core development. The underlying hardware is easily accessable while behaving functionally very close to the targeted system. The required toolchains for software development and debugging are almost identical to the GR712RC based implementation.

The structure is shown in Figure 3. The main building blocks are:

- The first FPGA-board emulates the SoC with CPU, memory controller and basic periphery.
- The second FPGA-board runs components that are placed in the programmable logic device in the base level design.

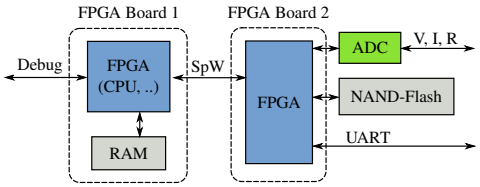


Figure 3. Two Commercial FPGA Boards Implementation

- Both boards are connected by a SpaceWire connection.
- Additional hardware like NAND-flash memory can be connected via standard pin headers to the second FPGA-board.

4. APPLICATIONS

All requirements on accommodation, reliability, environmental conditions etc. depend on the application, i.e., the specific mission and computing task to be performed. The choice between different implementations of the base level design as well as architectural considerations allow for adaptation to very different requirements. We provide three examples in the following. The main work items in this adaptation are in the board layout, the mechanical design, and test campaigns.

4.1. Application Scenario: Mini Instrument Controller

In this scenario we target a hypothetical instrument, that requires some computation and data storage, but only few external interfaces in comparison. The controller is not redundant and is supervised from a higher level controller, e.g., the satellite's central controller. The TM/TC port is included for completeness, but might be left unused in this scenario.

The minimal configuration possible with the base level design serves as a mini instrument controller shown in Figure 4. Physically the unit is composed of three *Printed Circuit Boards* (PCBs), marked by the dashed lines in the figure. Most digital circuitry, including CPU and FPGA, is placed on one PCB. Power conditioning and analog monitoring reside on another board. Connectors to the harness and line-drivers are placed onto a third board. These PCBs can be built reasonably small. An extrapolated configuration showed a size of about 100mm x 90mm. If stacked closely, they form a compact standalone box.

4.2. Application Scenario: Redundant On-Board Data Handling

This design is tailored to the specific requirements of the on-board data-handling systems of a small *Lower Earth Orbit* (LEO) satellite for a scientific mission. Figure 5 provides a block level schematic. The distinguishing characteristics are built-in redundancy and increased number of interfaces. Each half of the system is functionally almost identical to the previous design. To support the additional digital interfaces, a second FPGA has been integrated. Each half contains three cards, one containing the GR712RC base level design implementation. Power conditioning and additional analog interfaces have been added and are moved to a separate board, that carries all the required multiplexers and front-end electronics.

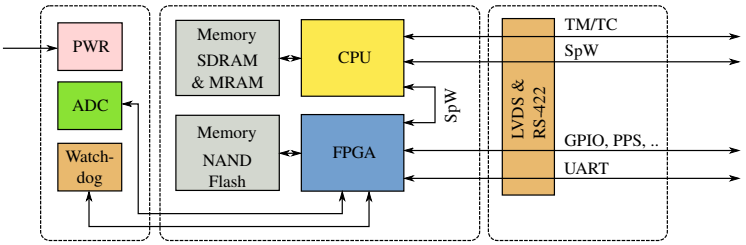
The two independent halves are incorporated into one housing using a backplane and daughtercard architecture. The common backplane is shown by the T-shaped area in Figure 5. The connections to the harness are concentrated on another board. This board also allows cross-strapping to be moved from the harness to the board, which allows simplifying the harness and eases integration. The board does not contain any active components. As a result, changes there can even be handled quite late in the development process.

4.3. Application Scenario: SpaceWire based Avionics

Using SpaceWire as an integral part of the base level design also allows building systems like the one shown in Figure 6. The unit previously described as the "mini instrument controller" can basically be put in the position of the on-board-data handling system of the satellite bus if the required interfaces are provided by additional RTUs. SpaceWire routers may be implemented in the programmable logic of the base level design or as separate units to allow for simplified reliability analysis.

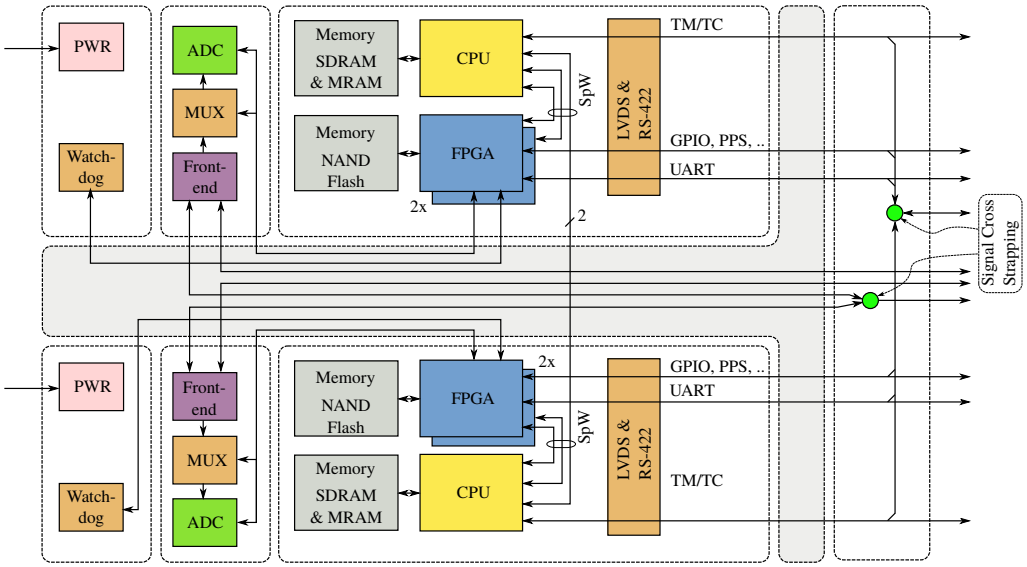
Connections to the power system and communication to the ground station are separated from the SpaceWire network. This simplifies FDIR. Using dedicated connections guarantees that no other systems can interfere with fault-isolation and safe-mode operations, thereby removing some possible faults that are hard to analyze and address in the design.

The router based design isolates external interfaces into separate RTUs. This simplifies the development process, in particular for complex specialized interfaces or new interface requirements. The new hardware can be functionally and – to a certain extent – electrically isolated from the core computing system. Besides fault isolation this also provides the opportunity to handle the new aspects in an independent development team or in a different organization which is a common scenario for scientific missions.



(Dashed areas are separate PCBs.)

Figure 4. Application - Mini Instrument Controller



(Dashed areas are separate PCBs.)

Figure 5. Application: On-Board Data Handling

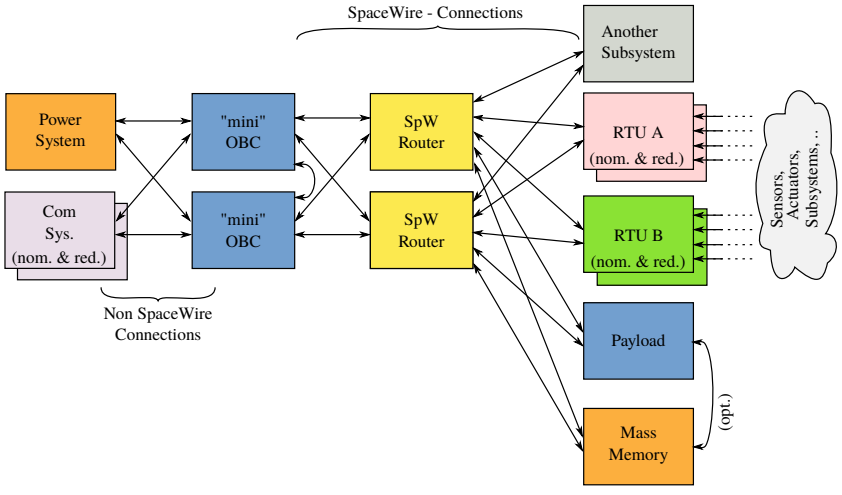


Figure 6. SpaceWire based Satellite Avionics

Table 2. Dependency of scaling different dimensions on different levels in the design

	Scaling Dimension				
Level	Reliability (w/o SEE)	Reliability (w/ SEE)	Redundancy	Computing Perf.	Cost
Part	X			X	X
Base Level Design		X			
Architecture			X	X	X
Software		X	X		

5. EVALUATION OF SCALABILITY

The two implementations and three application scenarios highlight the flexibility of the base level design and implicitly show the scalability provided by our approach.

The development effort spent on one design can efficiently be reused, and reshaped into each of the other systems, and other possible arrangements. Apart from the base level design itself, the IP-cores in FPGAs share a common framework and are often identical in different applications and implementations. This also applies to RTUs in the SpaceWire avionics scenario if an FPGA based implementation is chosen.

Table 2 gives a rough overview which dimensions of scalability depend on which level in the overall design. In the following, we analyze the potential for scalability more thoroughly starting with limitations, analyzing the architecture and, finally, the important reliability aspect that is tightly coupled to other dimensions of scalability.

5.1. Limits of Scalability

The designs of the circuit boards and the mechanical designs of the enclosures do not scale. They are tailored to a specific application as accommodation space, weight, and number of interfaces are driving the electrical and mechanical design. Thus, finding a common form factor does not work well. Assuming that a standard OBC is designed with the most common interfaces, this unit may be reused and RTUs may offer further interfaces.

5.2. Architectural Properties of Scalability

Scalability on the architectural level is achieved by various means, the major ones are:

- Hiding the details of implementation and hardware from the software through a HAL
- Platform independent and parameterized IP-cores
- Standardized and interoperable interfaces on IP-core level, e.g. AMBA and Wishbone
- SpaceWire provides a flexible basis for many topologies

These features allow for scalability in computing performance, available interfaces, as well as concepts for FDIR.

5.3. Scaling Reliability

Reliability depends on three major aspects in the design:

- Reliability of individual EEE-parts
- Data integrity
- Architectural redundancy and fault mitigation

The following three sections show how these aspects work and how they can be scaled.

5.3.1. Part Level Reliability

Increasing the reliability of a system by increasing the reliability of the individual components is typically achieved by testing and screening the components for faults and defects or by using more robust components. For example, environment influences due to vacuum and radiation can be addressed by hermetically sealed packages and radiation-tolerant parts.

The base level design can be implemented with parts of different screening and packaging options. This opens the possibility to choose parts that are either less expensive or offer higher performance, at the loss of screening/testing or tolerance to stressful environments.

This allows the requirements of a mission drive the selection of parts and results in a design, which can be tailored to the specific requirements.

We use the simplifying assumption that no component in the base-level design must have a failure for the design to work properly. Moreover, failures on links between the components are considered as failures of the respective components. Then, the reliability of the base level design can be estimated. Assuming that component $i \in \{\text{CPU, programmable logic, volatile mem.,}$

non-volatile mem., ADC, watchdog, PWR, board} has a reliability of r_i, the reliability of the base-level design is simply given by $r_{\text{base}} = \prod_i r_i$, where r_i is measured, e.g., by the probability that the component is available after a certain time.

Assuming that SEEs would have to be taken into account in this estimation, reliability would always be very low as, e.g., memories and flip-flops frequently suffer from bit-flips. Thus, mechanisms to ensure data integrity are discussed separately.

5.3.2. Data Integrity

Single Event Upsets (SEUs) in memories and digital circuits are a common phenomenon in space applications. All implementations of the base level design use some form of memory that is subject to bit-flips. Consequently, SEU mitigation is required. Through modifications in the software layer, the number of tolerable faults can be adapted to different SEU rates and to different levels of criticality of the data stored in the memories. Memories and GR712RC use *Error Correcting Codes* (ECC). For the NAND-flashes software-level redudancy is required and latch-up protection is available.

5.3.3. Architectural Level Reliability

Depending on the application's reliability and availability requirements, redundant systems are necessary. This can be implemented in different ways. In the on-board data handling application, two independent warm-redundant strings are used to provide redundancy, and almost continuous operation. This results in the reliability of a parallel system $r_{OBDH} = r_{\text{base}}^2 - 2r_{\text{base}}$ if the reliability of the mechanical separation onto daughtercard and backplane is neglected. Electrical effects are assumed to be independent due to the spatial separation.

In the instrument controller application a cold-redundant configuration is used, and FDIR is implemented in another system on a a higher level.

The SpaceWire avionics application uses redundant OBCs, and a massive amount of cross-strapped components.

From the hardware point of view, cold-spare capable interfaces are required if excessive wiring and interfaces must be avoided.

Most of the system's control over the FDIR process is implemented in software, and can therefore be adjusted to the specific needs.

In summary, we proposed a base level design for an OBDH unit facilitating IP-reuse and scalability in various dimensions. These aspects are thoroughly discussed and interdependencies are high-lighted. Three application scenarios show different instances of our architecture.

REFERENCES

1. S. Föckersperger, K. Lattner, C. Kaiser, S. Eckert, W. Bärwald, S. Ritzmann, P. Mühlbauer, M. Turk, and P. Willlemsen, "TET-1 a german microsatellite for on-orbit-verfication," in *4S Symposium: Small Satellites Systems and Services*, 2008, pp. 1–11.

2. S. Habinc, A. Sakthivel, J. Ekergarn, and A. Björkengren, "MASCOT on-board computer based on GR712RC," in *Data Systems In Aerospace (DASIA)*, 2013, session B5.

3. R. Jaumann, J. P. Bibring, K.-H. Glassmeier, M. Grott, T.-M. Ho, S. Ulamec, N. Schmitz, H. U. Auster, J. Biele, H. Kuninaka, T. Okada, M. Yoshikawa, S. Watanabe, M. Fujimoto, and T. Spohn, "A mobile asteroid surface scout (MASCOT) for the Hayabusa 2 mission to 1999 JU3: The scientific approach," in *Lunar and Planetary Science Conference (LPSC)*, 2013, abstract no. 1500.

4. SAVOIR Advisory Group, "SAVOIR: Reusing specifications to improve the way we deliver avionics," in *Embedded Real Time Software and Systems (ERTS)*, 2012, session 6c-1.

5. J. Lyke, D. Fronterhouse, S. Cannon, D. Lanza, and W. Byers, "Space plug-and-play avionics," in *AIAA Responsive Space Conference*, 2005, abstract no. 5001 (13 pages).

6. *SpaceWire-Links, nodes, routers and networks*, ECSS-E-50-12A 2003.

7. R. Lopez, G. Soragavi, M. Deshmukh, and D. Ludtke, "Knowledge management tools integration within DLR's concurrent engineering facility," in *IEEE Aerospace Conference*, 2013, pp. 1–11.

8. R. Aitken, G. Fey, Z. T. Kalbarczyk, F. Reichenbach, and M. S. Reorda, "Reliability analysis reloaded: how will we survive?" in *Design, Automation and Test in Europe*, 2013, pp. 358–367.

9. (2013) Gaisler GR712RC Dual-Core LEON3FT SPARC V8 Processor. [Online]. Available: http://www.gaisler.com/index.php/products/components/gr712rc

10. (2013) Microsemi ProASIC3 Series FPGAs. [Online]. Available: http://www.microsemi.com/products/fpga-soc/fpga/proasic3-overview