Forschungsbericht 2016-03

OBC-NG
Concept
and
Implementation

H. Benninghoff, K. Borchers, A. Börner, M. Dumke, G. Fey, A. Gerndt, K. Höflinger, J. Langwald, D. Lüdtke, O. Maibaum, T. Peng, K. Schwenk, B. Weps, K. Westerdorff

Deutsches Zentrum für Luft- und Raumfahrt
Simulations- und Softwaretechnik
Braunschweig

DLR Deutsches Zentrum für Luft- und Raumfahrt

Table of Contents

1. About This Document

OBC-NG is the abbreviation for **o**n-**b**oard-**c**omputer **n**ext **g**eneration – a project founded and made by the German Aerospace Center (DLR). The project goal is to provide the basis for future on-board computer (OBC) for space-missions. This document summarizes the conducted work, made in the DLR-project OBC-NG and its predecessor project "Software and Hardware Architecture for Re-configurable Computers" (internally named SHARC).

Document Overview

The actual technical report begins with Chapter 2, informing the reader about the motivation, the goals and the general requirements of this project and therefore serves as an introduction to the project. Further, the state of the art is reflected by similar on-board computer projects. After the introduction, Chapter 3 elaborates the design. The entire project is explained on the system, software and hardware level, to provide a deep understanding of the system and its concepts and solutions. For a deeper understanding, Chapter 4 states potential payload applications for the system. These applications have also been integrated in the project. Chapter 5 provides a compact overview of all software and hardware components, which have been integrated in the project. Insights, which have been gained while the project, are collected in Chapter 6. They are subdivided in hardware, software, implementation, application and evaluation issues. The review report closes with Chapter 7, providing a conclusion with a future outlook.

2. History – Assumptions, Studies, First Decisions

The project OBC-NG started at the beginning of 2012, internally named as SHARC. In 2013 we renamed the project to OBC-NG. SHARC was meant to be as a pre-studying phase. We defined what we would need, investigated what the current state of the art is and how our concept should look like. Many fundamental decisions had been made in this first year. This chapter will cover the results mainly of SHARC and should also deliver the reasons for the OBC-NG system concept, described in the OBC-NG Concept Chapter.

2.1. Motivation

In the upcoming years, the demand of on-board computing power for spacecraft is expected to grow steadily. This plus in computing performance is requested due to the increasing amount of payload data, which needs to be processed. A special focus resides on sub-domains like earth observation and space robotics.

On one side, the resolution of earth observation sensors is constantly increasing in all dimensions (spatial, temporal, spectral etc.). Due to a limitation in bandwidth, especially for low earth orbit satellites, certain pre-processing steps, like filtering, selection or compression of data, have to be conducted.

On the other side, robotic systems become more and more autonomous. Communication latency issues demand a growing autonomy, especially on deep space probes or rovers. However, this autonomy requires more complex control algorithms, which request more sensor data as input to derive a decision.

These two space exploration sub-domains reveal that the computing performance and the sensor advancement go hand in hand, making it necessary

to adjust them to each other. In contrast to the need of this plus in computing performance, strong general requirements of spacecraft can be identified. Requirements like reliability and durability have led to very conservative design decisions, mostly resulting insignificantly weaker computing performances than terrestrial technologies. Current redundancy concepts on subsystem level waste a lot of computing power. Many of the subsystems follow a cold or warm redundancy concept with a one to one mapping of identical hardware. This strict mapping forbids the usage of spare hardware of one subsystem in another one, binding an enormous amount of computing for no use.

Also problematic is re-usage of computational power, which is requested in one mission phase, in another. In case of a lander mission, the autonomous navigation while landing requires a huge amount of computing performance. The re-usage of that existing performance after the touch down for other purposes still represents a big challenge [1].

2.2. Application Requirements

The OBC-NG project targets computing intense domains with on-board evaluation of sensor data as one of the main challenges. This challenge ought to be mastered with commercial-off-the-shelf (COTS) components, in contrast to space domain common space-qualified parts. The inherent loss of dependability when using COTS components has to be compensated with new reliability concepts, mainly on software level.

There are two main reasons for on-board processing of big sensor data. Either the result of the evaluation of the sensor data is requested for the space vehicle to determine its current state, enabling it to react on the environment (e.g. navigation). Or, the amount of sensor data is exceeding the available bandwidth of the downlink to ground station, making an on-board pre-processing of the

sensor data necessary. As examples, ATON and on-board image processing are presented to offer practical insight in the application requirements [1].

2.2.1. ATON

Within the scope of the project, it had been the task to determine the requirements for an OBC to execute the ATON project (Atonoumous Terrain-based Optical Navigation). The ATON project addresses the autonomous landing on the moon by exploitation of optical navigation. Specifically optical navigation requires a huge amount of computing performance. The OBC-NG project targets the usage of COTS components with increased computing performance compared to space specific hardware. This plus in computing performance is required for the ATON project to succeed in a real space mission. Crucial for an optical navigation is the dependability of the system. ATON requires a hard real-time behavior of the system to retain the results of the tracking per frame within a certain amount of time .

2.2.2. On-board Image Processing

On-board image processing faces two major challenges. On one side, the huge amount of high- resolution image data creates an acquisition bottleneck due to limited downlink bandwidth and on-board memory. On the other side, a certain processing delay of the gained image data might weaken the expressiveness of the to be derived verdict. This becomes obvious when we consider a satellite supported weather catastrophe forecast system.

The only reasonable option is to process and evaluate the data where they have been sourced – on-board .Image processing requires moderate to high processing power. Therefore, OBC-NG seems to be a suitable platform for realization. To evaluate the capabilities of the OBC-NG project, the cloud detection application has been selected as an example of an earth observation

task.

When comparing the two example applications, ATON and cloud detection, it becomes visible that they both process image sensor data, hence still have completely different requirements. In ATON the complexity of the algorithm to navigate the spacecraft is in foreground. Whereas, the cloud detection demands the computing power for the huge amount of the to be processed image data. Therefore, the OBC-NG system requires a certain flexibility to target them both.

2.3. System Requirements

One goal of OBC-NG is to use commercial hardware and software for space applications, but we also wanted to exploit new developments in high-performance-systems. At the beginning of the project, we tried to get an overview about the current situation in hardware- and software development. We quickly realized that there is currently a paradigm shift toward massive parallel processing. With this situation in mind, we defined three main requirements for OBC-NG:

1. **Load-Balancing:** The utilization of all computing nodes in the spacecraft should be optimized. Application should run on several nodes in parallel.
2. **Redundancy:** Defective nodes should be replaced automatically in a running system. Working nodes should resume the applications of defective nodes.
3. **Re-Configuration:** It should be possible to configure the computer system for different mission phases with different applications.

From these main requirements, we derived requirements that are more specific. The requirements are partitioned into the categories application, operating system (OS), hardware, network and development environment.

We demand from the applications:

- to run on multiple threads,
- to run on different platforms,
- to minimize the network traffic and
- to export and import its state for backup and recovery.

We demand from the OS:

- to support multitasking,
- to run on different platforms,
- to support N-times redundancy,
- to boot fast,
- to encapsulate applications well,
- to support moving of the applications,
- to support load balancing,
- to monitor itself and
- to be controlled from the outside.

We demand from the network and the hardware:

- to be structured as a distributed system,
- to support different network topologies,
- to support quality-of-service,
- to support software sandboxing, monitoring and re-configuration

We demand from the development environment:

- to have a fixed development-flow for applications,
- to deliver a build- and simulation environment for applications,
- to support planning of the application-allocation on the nodes.

2.4. State of the Art

This chapter gives an overview about the available technologies on the market for OBCs for space. Furthermore, the rad-hard solutions will be compared to selected commercial solutions.

2.4.1. Hardware

The biggest gap between space-qualified and commercial parts can be found in the area of CPUs and microcontrollers. However, commercial electronic is sensitive to single-event-effects (SEE). Some of them, like SETs or SEUs, are temporary and can cause bit-flips in the data or the processing chain. Some of SEEs, like SELs, can also cause permanent damage. On a SEL, a parasitic thyristor is triggered, followed by a high current to the substrate [2]. Radiation hardened electronics for space is immune against the most SEEs by using special silicon-technology, special logic-cells or TMR.

Table 2-1 shows the most powerful rad-hard processors and on-board computers on the market in comparison to a commercial CPU for embedded systems.

Brand, Type	Performance	Description
Aeroflex UT700	233 DMIPS @ 166 MHz	Rad-hard, single-core, LEON3FT based microcontroller with SDRAM memory controller, Spacewire, Ethernet and MIL-1553 interface
Aeroflex GR712	140 DMIPS / 200 MFLOPS @ 100 MHz	Rad-hard, dual-core, LEON3FT based microcontroller with SDRAM memory controller, Spacewire, Ethernet interface

Atmel AT697F	86 DMIPS / 23 MFLOPS @ 100 MHz	Rad-hard, single-core, LEON2 based microcontroller with SDRAM memory controller and PCI
Maxwell SCS750	1800 MIPS @ 800 MHz	The Maxwell board is a complete computer solutions consisting of three PowerPC CPUs. The CPU-outputs are connected to a Microsemi FPGA which makes majority logic on the inputs. Additionally SDRAM, EEPROM and a second Microsemi FPGA are placed on the board.
Space Micro Proton 400k	5760 MIPS @ 1,2 GHz	Board computer solution consisting of two PowerPC e500 CPUs. The error-correction is done by the patented method "time-triple-modular redundancy": Both CPUs running in parallel. When the CPUs deliver different results, the processing will be repeated.
Space Micro CHRCE Space Processor	5000 DMIPS for both CPUs @ 1GHz	Radiation tolerant microcontroller board equipped with a Xilinx Zynq-7000 SoC, containing a dual-core ARM Cortex A-9, which is runnable with up to 1 GHz frequency.
Freescale i.MX6	ca. 4.000 DMIPS / 2 GFLOP @ 1GHz + 4 GFLOP for the NEON-co-processor	Commercial microcontroller family based on a dual core ARM Cortex A9 CPU.

Table 2-1 List of rad-hard processing / computer solutions

The commercial processor (i.MX6) is more than ten times faster, compared to

the fastest rad-hard processor (UT700). The complete computer-solutions come close to the performance of the commercial processor, but are less flexible.

For performance critical applications, CPUs are mostly to slow. Here DSPs, GPUs and FPGAs deliver more performance and a better performance to power ratio. Table 2-2 shows an overview over the best performer in this area for space and for commercial applications. The power consumption of the FPGA strongly depends on the application. In general, the performance to power ratio of FPGAs is better than of GPUs, because algorithms can be efficiently pipelined. So the access to external memory can be minimized.

Brand, Type	Performance	Description
Microsemi RTAX4000D	5 GFLOPS @ 125 MHz	antifuse, rad-hard FPGA
Xilinx Virtex 5Q	40 GFLOPS @ 400 MHz	rad-hard FPGA, SRAM based with single event upset (SEU) immune memory-cells for configuration
Xilinx Ultrascale Kintex KU115	ca. 1.4 TFLOPS @ 500 MHz	commercial FPGA, SRAM based, 20nm silicon process
Nvidia GeForce GTX 980	ca. 5 TFLOPS @ 1.1 GHz	commercial GPU, power consumption is around 150 W
Texas Instruments TMS320C6670	ca. 80 GFLOPS @ 1.2 GHz	commercial DSP with 4 cores

Table 2-2 Selection of FPGAs, GPUs and DSPs, available on the market

The GFLOP-performance of Table 2-2 is only a weak indication for the real-world performance. Memory bandwidth is also an important factor. Though, it depends on the application and the implementation how much of the theoretical computing power can be exploited.

Asano [3] did evaluate the performance for three image processing algorithms

(2D filter, stereo-vision and K-means-clustering) on CPU (Intel quad-core Q6850), GPU (Nvidia GTX 280) and FPGA (Xilinx Virtex 4 XC4VLX160). For the 2D filter, the GPU was the fastest, up to 5 times faster than the FPGA and up to 10 times faster than a CPU depending on the filter depth. For the more complex algorithms, the FPGA was the winner, with up to 10 times more speed than GPU or CPU, which are on a similar level.

Jones [4] came to different results. He presented a benchmark test for FPGA and GPU with focus on high performance computing (HPC). Aside from the random pattern generation, the GPU was always much faster than the FPGA.

Not so long ago, the usage of GPUs for embedded systems was unattractive, because of the high power consumption and the need for active cooling. However, recent developments, mainly driven by mobile media-solutions, brought some interesting devices on the market. Nevertheless, FPGAs will stay the first choice for space applications because of the unbeatable flexibility and the best performance to power ratio.

2.4.2. Network Technology

There exist a wide range of network technologies and protocols for different applications. Use cases might be industry automation, entertainment devices or high-performance computing. For OBC-NG, we investigate network protocols suitable for space applications. We especially elaborated the automotive and avionics sector, because of the similar requirements regarding security and reliability. Mass and required space had been also criteria. Table 2-3 shows the result of the investigation.

Network	Data Rate	Topology	Access Management	Field Tested	Redundancy

Network	Data Rate	Topology	Access Management	Field Tested	Redundancy
TTEthernet	>1 Gbit/s	switched and point-to-point	TDMA	NASA Orion, several space projects	by redundant connections and switches
AFDX	100 Mbit/s	switched	event based with bandwidth limit	Airbus A380, Airbus A400M	by redundant connections and switches
Flexray	10 Mbit/s	bus, Star, combination of both	TDMA	automotive	dual redundant connections / channels
TTCAN	1 Mbit/s	bus	exclusive or prioritized time slices	automotive	no redundancy
SpaceWire	400 Mbit/s	switched and point-to-point	event based	multiple space projects	no redundancy
TTP/C	25 Mbit/s	bus, Star, combination of both	TDMA	Boeing B787, Airbus A380 and other aircrafts	dual redundant connections
MIL-STD-1553B	1 Mbit/s	bus	event based	multiple space and avionic projects	redundant bus-structure
Firewire 1394b	800 Mbit/s	chain, tree	bandwidth lime (isochronous, asynchronous)	in several military projects	no redundancy
Fibre Chan-nel	10 Gbit/s	switched	event base (QoS possible)	Boeing F/A-18, Lock-heed Martin JSF	no redundancy

Table 2-3 Network technologies suitable for space

The available data rates show a wide range from 1 Mbit/s to 10 Gbit/s. The search for a high performance network demands a minimum bandwidth of 100 Mbit/s for our purposes. So Flexray, TTCAN and MIL-1553B do not need to be considered. Another requirement is the robustness of the system. If the network architecture already contains any kind of redundancy, it will not be necessary to compensate already covered failures on system level. In addition, switches and end-points must be adapted to fulfill the space requirements in reliability. Robustness can also be obtained by a reasonable network topology. In networks with multiple ways to an end-point, disconnects could be easily compensated by changing the routing. FireWire has no redundancy included, nor a topology that allows re-routing in a case of disconnection. Therefore, the FireWire does not fit to our needs.

If real-time is a requirement, the choice would be TTEthernet, AFDX or TTP/C. All these networks have been used in space or avionics in the past. Nevertheless, TTP/C is limited in bandwidth and supports only bus or star topology. That is why the Ethernet-based networks seem to be the most promising technologies.
If quality of service is not a hard requirement, SpaceWire is still a good choice. Compared to other networks the protocol is fast and simple. It has a long heritage in the space community and the robustness of SpaceWire had been proven in several space projects.

2.4.3. Operating System

An evaluation regarding OSes for on-board computers had been conducted within the SHARC project. The outcome of this evaluation had been taken as basis for the decision making regarding the OS for OBC-NG project.

Interpreting the results reveals that none of the OSes can convince in all criteria. The recommended OS of the SHARC evaluation is RODOS (Real-time Onboard

Dependable Operating System). Two aspects influenced this decision. On one side, the results of the most important criteria ranked RODOS as the leading operating system. On the other side, RODOS is a partly in-house developed operating system. There is free access to the source code and a RODOS co-developer is available in the department.

2.4.4. Similar On-Board-Computer Projects

Other space agencies and companies run similar projects in search for more processing power. We will introduce three of them in this chapter. Each project has its own approach for more flexibility and performance compared to classic on-board computers.

Singapore's X-Sat Satellite

A high performance-processing cluster with COTS components had been built for the Singapor X-Sat project [5]. Because of redundancy reasons, the system consists of two identical halves. Each half contains 10 high-speed processing nodes (PNs). The heart of each node is Strong-ARM CPU made by Intel. Each CPU has its own 64 Mbyte SDRAM memory (see Figure 2-1). The PNs are grouped around an Actel-FPGA. Defect or unused PN can be disabled by the FPGA.

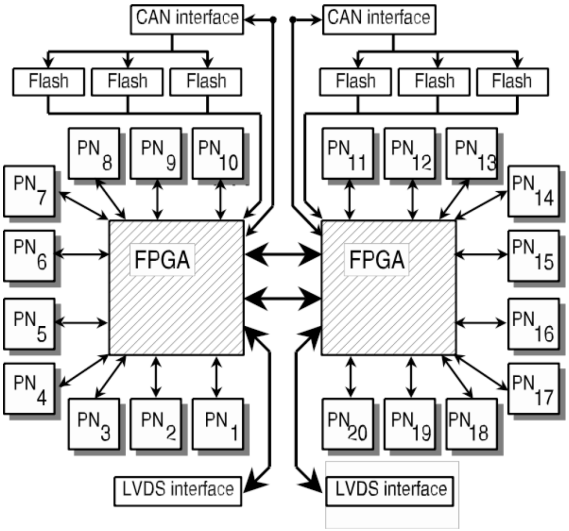

Figure 2-1 Processing cluster structure of the Singapor X-Sat [5]

The FPGAs provides the network-interface, monitors the PNs and provides the boot-software for the PNs. For this purpose, the FPGA is visible inside the PN memory space. The network inside the FPGA is built as a ring. Each PN gets a fixed time-slice for communication. Data can be sent as broadcast or as point-to-point transmission. The two FPGAs work in master-slave configuration.

Embedded LINUX runs as operating system on the PNs. Dedicated drivers had been developed for the network. Because of the use COTS components for the PNs the software is responsible for error-detection-and-correction (EDAC) of the PN memory. The software generates error-codes for selected memory-areas and checks them periodically.

The start-command for the application comes from the CAN-bus. Thereupon the FPGA appoints a PN as master. The master will activate additional PNs and assigns tasks to them. The data handling of application is managed by the master.

Processing Architecture on Iridium NEXT

For the Iridium NEXT satellite system, now announced for 2017, the company SEAKR got the contract to build a flexible and powerful on-board-computer for handling the communication [6]. For Iridium NEXT a fleet of 66 satellites are planned. This high number of satellites and components are made in almost series production. Therefore, SEAKR decides to use a mix of rad-hard and commercial components.

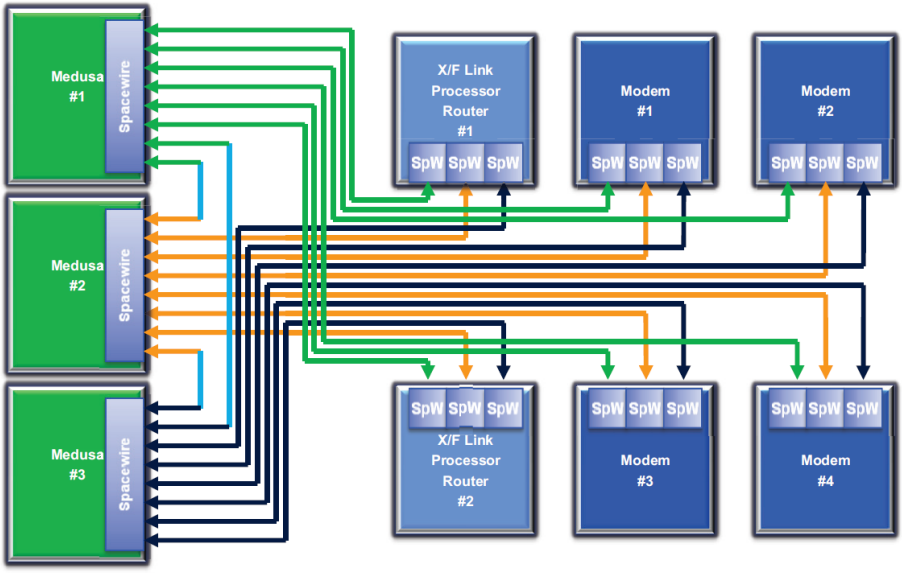

Figure 2-2 Computer architecture of Iridium NEXT satellite [6]

The Iridium NEXT board-computer consists of three CPU-boards with commercial PowerPC CPUs, which are called "Medusa" (see Figure 2-2). Four FPGA boards work as modem for modulation, demodulation, routing and CODECs. Each FPGA boards consist of three rad-hard Xilinx Virtex 5Q FPGAs. Each FPGA-board is directly connected to each CPU board via Spacewire. The CPU-boards manage the processing on the FPGA-boards. A dedicated middleware is responsible for

the dynamic assignment of the tasks.

FPGA configuration and software can be updated in space. Therefore, the operation of the satellite can be adapted to future requirements. The redundancy concept for the on-board-computer relies on the high number of connections between the sub-systems. So the computer can still work, after multiple components failed. Only the performance will drop.

High Performance Dependable Multiprocessor

Honeywell developed an on-board-computer for the NASA millennium program called "dependable multiprocessor" [7] [8]. The project was started in 2004. The performance should be increased by the use of high-performance COTS processors, so the project-goal was the same as OBC-NG. A technology readiness level of 6 was reached in 2009. An in-flight verification, planned for 2007, was canceled.

Figure 2-3 shows the hardware structure of the dependable multiprocessor. Two redundant rad-hard system-controllers manage and monitor the data processors. The system-controllers also communicate with the spacecraft. Over a redundant high speed network, the COTS based data-processors can be accessed. Central element of the data processor is a PowerPC-CPU of the type 7447 and an AltiVec coprocessor.

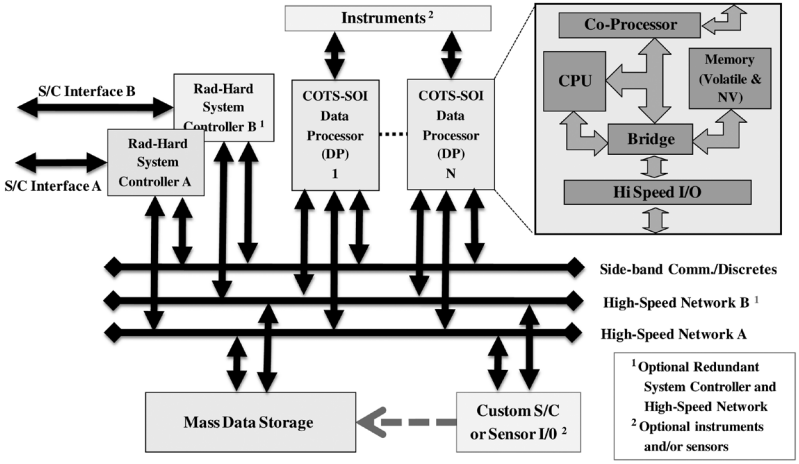

Figure 2-3 Hardware structure of dependable multiprocessor [8]

A middleware, running on system-controller and data processors, handles the jobs and failure detection. The middleware also provides a platform independent application interface (API). On the data processors, Linux was chosen as operating system.

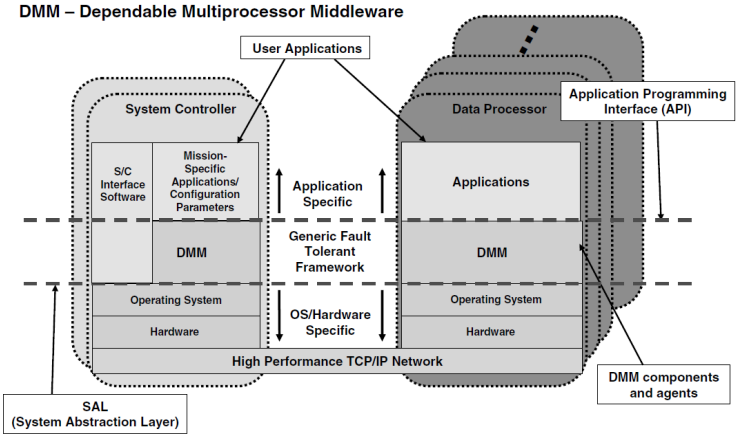

Figure 2-4 Middleware structure of dependable multiprocessor [8]

The investigation of the dependable-multiprocessor project shown, that SEU and SET sensitivity is the only major drawback of COTS components. With the use SOI technology single-event latch-up (SEL) issues can be avoided.

Failure management is done in a complex approach on several levels of hardware and software.

2.4.5. Trends in High-Performance Computing

During our investigation about the state of the art, it turned out, that SHARC / OBC-NG touches well known problems of the HPC and cloud-computing domain. Therefore, we took a deeper look into approaches and solutions of these two domains.

The physical limits of the silicon technology got obvious in the last years. Over decades with every silicon technology step the transistor size has been reduced and the frequency has been increased. This exponential increase in performance is well known as Moore's law. However, since approximately 10 years the maximum frequency seems to have limit of 3-4 GHz, observable on desktop x86-CPUs. To keep up with Moore's law the processor-makers have to integrate more CPUs on chip. But to exploit the number CPUs, the application software has to be divided into multiple parallel running threads. Classic software engineering methods struggle at this point. Scalability is often a problem. Not optimized software may run slower on many CPU cores than on a few cores.

Multi-processor systems usually have a common main memory. Information between software threads can be shared by using the same memory. Synchronization mechanisms have to be used to assure the consistency of the information. In multi-processor systems without shared memory, software threads can messages to other threads.

In high-performance computing, a paradigm shift to massively parallel software has already taken place. High-performance applications are optimized for parallel processing since years. Sequential programming became less important. In embedded systems and desktop programming, this paradigm shift started recently. The way to massive parallel processing does not touch only the application, but also the operating system. One example of a cloud-based operating system is Parallax [9] [10]. Parallax did influence the OBC-NG concept.

In current cloud-based computer systems, the applications are handled by the middleware, often structured into several levels seating on the top of the operating system. In Parallax the application management was reduced to the minimum and was integrated into a network-centric operation-system. Crucial functions of the operating system are encapsulated in self-managed units. These units are called DIME (Distributed Intelligent Management Element). They are responsible for the failure, configuration and performance management.

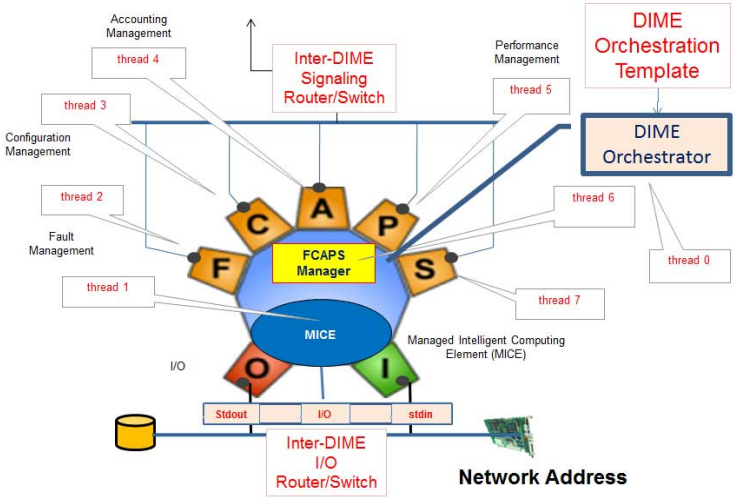

Figure 2-5 DIME structure in Parallax [9]

A central orchestrator organizes the DIMEs. This orchestrator assigns the DIMEs to the available computing-nodes. When a node has free resources, the orchestrator takes a new DIME out of the pool and gives it to the node. On the other side new DIMEs are put into the pool. Several DIMEs can be active on one computing node. The DIMEs control the requested performance by themselves.

The DIMEs can interact via messages. Every DIME has special input and output threads for this purpose. The messages are addressed to the unique ID of a DIME. The message routing is part of the OS.

3. OBC-NG Concept

The OBC-NG concept relies on the exploitation of high-performance COTS parts and targets shortcomings like reliability with new concepts on the software level. The system itself is a reconfigurable heterogeneous cluster with various tasks, like data processing, system management and interface operations. Within this chapter, the entire OBC-NG system is elaborated in detail. The content is mainly based on the paper: *OBC-NG: Towards a Reconfigurable On-board Computing Architecture for Spacecraft* [1], which can be used as a supplement.

3.1. System

Within this subsection, the high level system design is elaborated. Included are the top-level requirements, the system approach and the system design. An additional focus is set on the reconfiguration of the entire system, as it represents a novelty.

3.1.1. Top-level Requirements

The OBC-NG system is mission-independent but targets potential future space-mission requirements. Among such requirements is the increasing demand regarding more computational on-board resources of the spacecraft. This request cannot be solved with the current space-qualified components and their waste of computational resources because of one-to-one mapped warm and cold redundancy. Additionally, computational intense tasks can have different requirements. Control tasks often need to be finished within a certain time interval, the more computational performance is available, the more advanced control algorithms can be designed. On the other side, pure payload processing, for example filtering sensor data, can challenge the on-board computer, also if there is no strict deadline. An example for an application, which offers both scenarios, is a navigation and guidance system. Those systems are mostly based

on complex computer algorithms and heavy image data processing [1].

3.2. System Approach

The system is comparable to a heterogeneous embedded cluster. Different computing devices, like CPUs and FPGAs, act as nodes, which are interconnected via a switched network with each other. For the future, it is planned to integrate also embedded GPUs or DSPs. Data and task parallelism can be exploited to distribute and re-distribute the computational payload evenly on the system. Whereas, task parallelism takes place on system level and data parallelism on node level. A re-distribution is a reconfiguration of the system and the reaction on special incidences, like a malfunction of a node. Focused is also the usage of COTS components as nodes of the system to improve the overall computing performance. However, new techniques are necessary to generate the required reliability of the entire system, without applying the common one-to-one mapping of computational devices.

The distribute nature of a cluster demands a reliable and efficient interconnection among the different nodes of a system. Therefore, the network of OBC-NG can be planned up to a fully meshed network, interconnecting each node with all others, to avoid single points of failure. To increase the efficiency of the entire system, the specification of the interconnection of the nodes needs to match the nodes performance capabilities. Therefore, a network has to be preferred instead of a bus-architecture, as the bandwidth of networks is higher than comparable bus topologies [1].

3.3. System Design

Nodes

Figure 3-1 reveals an exemplary structure of the OBC-NG system. The amount of nodes and their class is variable, offering great potential regarding scalability.

There are two main classes of nodes, when investigating the functional level. Nodes can be either Processing Nodes (PNs) or Interface Nodes (INs). INs are in charge of the connection of peripherals, like sensors or memories, to the system. PNs take care of the processing of data and additionally manage the system.

A PN class node holds a Router (R) and a Main Processing Unit (MPU). The Router represents the gateway to the network and is hosted by an extra FPGA. The MPU can be supported in its task by a co-processor. This co-processor might be an FPGA, an embedded GPU, an DSP or any other computing device. Within the current prototype the co-processor is limited to an FPGA.

The IN class node hosts a Router and a less powerful micro-controller, compared to the MPU of the PNs. Its main task is to expose the peripherals to the system [1].

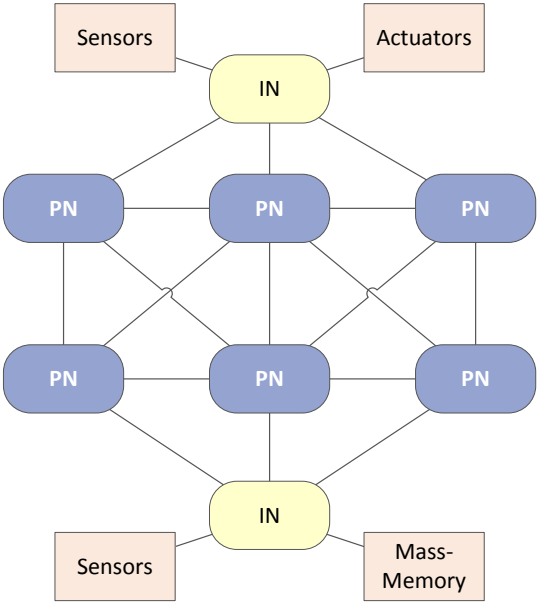

Figure 3-1: Simple example of an OBC-NG system [1]

Roles of Nodes

The possible role of a node is depending on its class and the functional task. Due to the re-configurable nature of the OBC-NG concept, roles can change within runtime of the system. We investigate the different roles, depending on the class of the node.

A PN class node can be Master (M), Observer (O) or Worker (W). The distribution of the roles over all PNs of the system is dynamic and depends on the current configuration. This also counts for the Master of the system, which has to control and monitor the system. The Master is an application, which monitors the entire system to be able to react accordingly. A reaction of the Master is necessary if a failing node is not able to resolve the issue on its own. A reconfiguration of the system is most likely the result. Currently, the reconfigured system lacks a node after this, but it is planned to integrated methods to reset and revive faulty nodes to integrate them back into the system in case this is possible. If the Master node is not fully utilized with its Master role, it can additionally process data and therefore serve as Worker, too.

To increase the reliability of the entire system the Observer role is given to two more PNs. The primary Observer investigates the behavior of the Master, whereas the Master checks on the Observers (in addition to the IN roles and Workers). The secondary Observer investigates the primary Observer and the Master. In case the secondary Observer detects an irregular behavior of the primary Observer, it informs the Master to let it decide on the consequences. However, in case one of the Observers realizes an irregular behavior of the Master, a reconfiguration of the entire system is triggered via a broadcast message.

The last role of a PN class node can be the Worker. There is no management task involved with this role, only the conduction of demanded tasks. The Workers are often organized in a client-server fashion, with multiple client workers and one server worker that gathers the information. Nevertheless, this is

depending on the type of parallelism, if it is task- or data-based, and resides in the hands of the application developer.

An IN class node can have the role of a Storage (S) or an Interface (I). In case of a Storage node, the IN is connected to a mass memory. A mass memory is required for housekeeping and scientific data. However, specialties arise in this case, like checkpointing services, which are in focus in a subsequent section. On the other side, the IN offers the connection of sensors to the network and therefore establishes the connection among the sensors and the Workers on the PNs [1].

3.3.1. Reconfiguration

Within the current and common spacecraft development, reliability issues are often solved by warm or cold redundancy. Additionally, subsystems are designed and implemented as independent units within the overall spacecraft system. Therefore, the redundant parts of each subsystem are one-to-one mapped to each other. In case of an issue, the redundant part takes over and continues the job. But it is not possible for one subsystem to take over a job of another subsystem. Therefore, a lot of potential computing power is wasted and loads of redundant parts are carried into space, which might never create any benefit for the space mission.

The OBC-NG concept breaks with this strict one-to-one mapping and introduces the one-to-many mapping. Spare nodes can be used to take over the jobs of faulty ones, which get separated of the system. Additionally, it is planned to determine their status, when possible fix them and re-integrate them into the system.

Addressing the stiffness of current approaches, another problem is targeted by the reconfiguration mechanism of the OBC-NG concept. Some space missions

have huge requirement differences when a mission phases changes. This becomes very drastic in autonomous lander missions. The spacecraft needs a huge amount on computing power to elaborate the sensor data it receives while landing, to instruct the control mechanism of the entire spacecraft. When the lander reached the surface, different tasks await it. There, the available computing power could come in very handy too. The OBC-NG concept would trigger a reconfiguration of the system from landing to exploration configuration, exploiting its potential computing performance.

Considering these two scenarios, a mission phase change and a failure event, the reconfiguration mechanism conducts planned and unplanned autonomous reconfigurations. The planned reconfiguration is performed on mission phase changes to allocate existing computation performance for new task. The unplanned autonomous reconfiguration takes place whenever the Master node realizes a severe failure in the system or an Observer realizes that the Master is not reachable.

In both cases, a decision graph is used to determine the next applicable configuration of the entire system. A simple example graph is displayed in Figure 3-2. The nodes of the graph represent configuration states and the edges are transitions from one configuration state to another, protected by guards. A guard is a specific error that occurs or a mission phase change event. The current prototype can detect errors only on node level and initiates the state transition regarding the specific node error. Reconfigurations can be conducted until a state is reached in which no valuable configuration can be reached anymore. Then the system takes the edge to a safe-mode, which is reachable from any state in the graph. The graph itself is cycle-free and its size can grow fast with the amount of nodes in the system as well as with the failure detection granularity (Currently the prototype detect failures only on node level, but the detection on sub-node and task level would increase the size of the graph by magnitudes) [1].

The OBC-NG concept considers a heterogeneous cluster as underlying hardware. The usage of FPGA technology is common in the space domain, but other technologies might be possible soon, like embedded GPUs. Currently, if an FPGA is faulty another FPGA has to take over and continue the job of the faulty one. OBC-NG wants to break this barrier by enabling the potential of task morphing to other technologies. It is of course of interest for the fulfillment of the mission goals to utilize a spare FPGA, in case another fails. But if no FPGA resources are left, a morphing of the task to a spare CPU to continue the mission can be an acceptable solution. This morphing is currently under investigation, whereas high-level synthesis tools, hardware/software co-design approaches or translators might be useful.

In the current prototype, a heartbeat monitoring mechanism is integrated to determine the healthiness of a node. This mechanism is a pull request with the Master as the initiator against all nodes and the Observers against the Master. It is planned that nodes check their current states regarding errors. If a node realizes that it is in an erroneous state of which it cannot leave, it informs the master via a message about its state. The Master then triggers a reconfiguration of the entire system. Additionally considered is, that nodes expect a certain communication of other nodes and if they do not receive the same, they inform the Master.

The reconfiguration itself is triggered by a broadcast message of the highest priority. While the reconfiguration process, nodes have to stop the transmission of messages via the network until the system is reconfigured. All binaries and bit stream files reside on each nodes internal storage to keep the payload on the network as low as possible and the reconfiguration time short. Each node just gets the information about the new configuration ID via the broadcast and searches the decision graph for the state to configure itself accordingly. In addition to that, the routing tables of the routers are also updated to match the new configuration.

Problematic can be when tasks that are migrated from one node to another require a specific start state. This can be achieved with the checkpointing service, which stores such states on a regular basis and forwards it to the current node with the task running after reconfiguration. That mechanism represents the warm redundancy concept in OBC-NG [1].

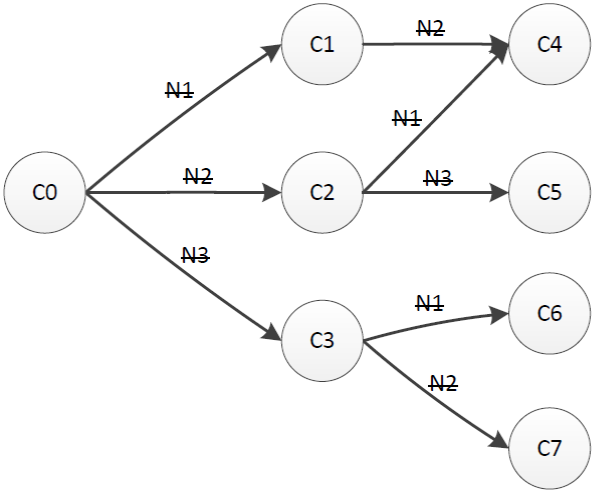

Figure 3-2: Simple example of a decision graph to mitigate node failure. Cx denotes the configurations and ~~Nx~~ the failing nodes.

3.4. Software Requirements

The important functional requirements that influence the software architecture are listed below.

- The OBC-NG software system shall be able to execute all necessary software on-board of a spacecraft. This includes avionic systems, command and data handling, payload control, on-board data processing etc.
- Since OBC-NG is a distributed system, the software must support

distributed systems and the required communication services as well as parallelized applications.
- The system architecture provides special purpose co-processors like FPGAs that are connected to the MPU of the Processing Nodes. These co-processors need to be accessed by the software system on the attached MPU.
- One of the basic concepts of OBC-NG is the use of reconfiguration to change the system for different mission phases and to mitigate errors. Hence, the software needs to support reconfiguration of hardware and software.
- The software system needs to provide an environment in which applications can be executed.
- Services to control and monitor the system, i.e. master node functionality, shall be implemented in software.

The primary architecture and quality goals of the software of OBC-NG are:

- **Performance:** The system shall achieve >10 GFLOPS. The reconfiguration shall be completed within 5 seconds. Real-time tasks that are not migrated during reconfiguration should not be interrupted for more than.
- **Reliability:** The system shall be able to recover from consecutive node failures by migrating important tasks to functioning nodes. Two consecutive node errors will be demonstrated. During the project OBC-NG, only complete node failures are considered. Single event upsets (SEU), which can be caused by ionized or electromagnetic radiation, are not considered in this phase of the project.
- **Availability:** The system shall always be responsive to commands from ground control as long as at least one node, which is able to perform the master functionalities, is operable.
- **Security:** Since the system is later deployed to a spacecraft, the only interface to the outside will be the Telecommand/Telemetry Interface.

Thus, no special security measures need to be taken within the system. The prototype will only be operated in local laboratory networks. Again, no security requirements exist to prevent unauthorized access.

- **Testability:** To ensure a high quality of the software systems for further development, all software components explicitly developed in this project that are not considered as prototypes shall have a high test coverage of at least 60 % (line coverage).

3.5. Software Architecture

The software architecture is a layered structure as depicted in Figure 3-3.

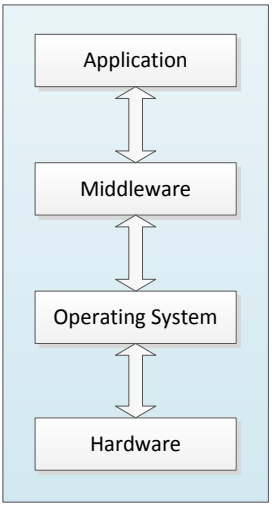

Figure 3-3: Basic software architecture of OBC-NG [1]

3.5.1. Operating System

The OBC-NG concept offers a very high scalability, especially in the amount of attachable nodes.

When a certain number of nodes is exceeded, the transition from cluster to grid

computing is conducted. It is not expected that this will happen in the near future, especially not in the embedded space domain. But the general trend in this direction has an influence on the development of operating systems. They become more light weight per node and offer greater possibilities for node specific configuration. This comes in handy when we investigate the I/O functionalities of a Linux OS as example, which are never used on a OBC-NG system, as the communication takes place via network [1].

An evaluation of commercial and academic operating systems revealed that there is no suitable operating system on the market, which covers all major requirements of OBC-NG. Especially the multi-core support for operating systems in the embedded systems domain is broadly neglected. Additional requirements, like stability, real-time capabilities and synchronization can only be reached by severe changes in the operating system.

Additionally, some envisioned applications with high demands to already available libraries, cannot be easily ported to a newly developed or modified minimal real-time operating system.

Therefore, two operating systems have been selected for OBC-NG. PetaLinux, a Linux variant, will be used for complex applications, to utilize the large set of available (3rd party) libraries. For time-critical applications, the minimal real-time operating system RODOS has been selected and adapted to Zynq multi-core board [1]. The RODOS is not multi-core ready, as the priority for a re-design of the network protocol was higher, and no specific drawbacks expected when RODOS runs on a single core with shared memory.

3.5.2. Middleware

The OBC-NG middleware is designed as a layered architecture. It consists of network protocol, Tasking framework and management layer to offer communication services, application task services and management as well as

monitoring services (see Figure 3-4). The OBC-NG middleware uses message-triggered and event-triggered mechanisms for task execution and management. The middleware supports the PetaLinux and the real-time operating system RODOS for the consideration of two aspects, i.e., some applications rely on third-party Linux libraries and some applications require hard real-time abilities of RODOS. The aforementioned features including management, monitoring, reconfiguration and model-based development are implemented in the OBC-NG middleware [1].

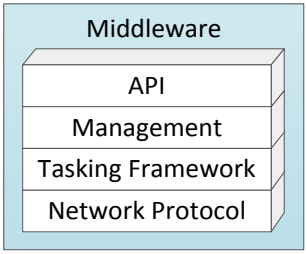

Figure 3-4: Structure of the middleware of OBC-NG

3.5.3. Network Protocol

The network layer is responsible for the message communication among nodes in the distributed on-board system. The network layer, which is shown in Figure 3-5, incorporates network protocol, network connector, underlying protocol, event handler and timer service. The network protocol is the core part of network layer, which transmits and receives messages of different transmission types to/of the network. The network connector is an abstract layer for the network protocol to transmit and receive messages through the underlying protocol. The event handler handles the received data and is triggered by the network protocol. The timer service is to offer timer functionality from hardware.

Currently, the underlying protocol supports Ethernet with UDP/IP. The next step

is to integrate the SpaceWire Protocol since it is widely used in the space domain. SpaceWire is a network connection that is low-cost, low-latency, full duplex and it is based on point-to-point serial links and uses the packet switching wormhole routing. The OBC-NG network protocol does not only support the transmission of unreliable and reliable messages but also large-size messages. Moreover, the subscription and broadcast mechanisms can be realized by using the network protocol. Furthermore, it is also designed to support monitoring, error detection and reconfiguration on higher layers.

With the STUP (Serial Transfer Universal Protocol), efforts have already been conducted to define and integrate a light-weight protocol on top of the SpaceWire protocol. This protocol describes the format of simple read- and write-commands and their message structure.

The OBC-NG network protocol is a more advanced approach compared to STUP. Packets that exceed a certain maximum payload size per packet are split-up in multiple packets. An acknowledgement mechanism is integrated to take care about the transmission, with additional resend notifications. In addition, error notifications are covered. It is possible to use push as well as pull transmissions mechanisms. We want to present the OBC-NG protocol at the SpaceWire Conference 2016.

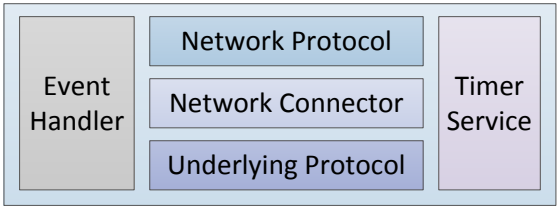

Figure 3-5: Network layer structure

3.5.4. Tasking Framework

The Tasking framework offers applications task services. The Tasking concept comes from RODOS of DLR. It provides communication and scheduling for task-based applications. To use the Tasking framework, application developers need to divide their algorithms into smaller chunks. These tasks can then be distributed on several nodes or cores on a multi-core CPU. In the Tasking framework, a task has three actions, i.e., consuming information, performing computations and producing information that can be used as input message by other tasks. Computations are atomic which means that through the atomic computation, the state of the system will be updated or a value will return as an output. The task computation is triggered either by an event or by the fact that all required input data had been collected. The results of computations are information, which can be used by other subsystems or modules. The Tasking framework can be used for distributed and shared-memory system architectures. The communication media are messages and events. The workload partitioning and task mapping need to be realized explicitly. The Tasking framework offers thread management and synchronization. It has been used in the onboard software for several space missions at DLR. Currently, the Tasking framework can run on both OS, Linux and the real-time operating system RODOS [1].

3.5.5. Management Layer

The management layer offers management, monitoring and reconfiguration services for nodes in coarse granularity and tasks in fine granularity. The management layer has five management tasks: monitoring, reconfiguration manager, reconfiguration service and checkpointing service. For the project OBC-NG, nodes can be either of the type PN or IN. Computation and management tasks run on the PN. Thus, the role of PN can be Master (M), Observer (O) or Worker (W). As the onboard system includes various peripheries such as sensors, actuators, instruments and mass storage, the IN is the connection part between the network and peripheries. Therefore the role of IN can be Storage (S) or Interface (I). The IN is also responsible for the management

of data subscription lists, i.e., the periphery sensor will send acquired data to the tasks, which are registered in the subscription list of this periphery sensor [1].

3.5.6. API Layer

The Application Programming Interface (API) is provided in the OBC-NG project with communication as its core service. The message passing is handled by the Tasking framework, which is accessible via the API. Application developers are requested to decrease the granularity of their tasks, to allow a better load distribution over the entire system and its nodes. Important is to define the interface and the internal state of each task. The interface, input and output, is important for the communication and the internal state is necessary for the checkpointing. Tasks can then be triggered by a timer event or an input event.
The application developers are free to decide on the level of parallelism. The task level parallelism can be tuned by the size of the tasks. But the communication overhead and the amount of available multithreading capability of the entire system has to be considered. The data level parallelism can be exploited on node level. OpenMP is a commonly used framework in that context, but future approaches could also use a direct memory access from the FPGA to the CPU RAM to let the FPGA process on data in parallel with the CPU. Therefore, a next step within the development of the API would be an interface to specific co-processors, like FPGAs [1].

3.6. Hardware

In principle, the OBC-NG is not bound to any specific hardware. We only defined a rough hardware structure for the processing nodes and for the network. We further looked for the most promising hardware-platform, where we could verify the first implementation of the OBC-NG software.

3.6.1. Processing Node Hardware

Each processing node in principle consists of a CPU, memory, router and an optional coprocessor, as shown in Figure 3-6. FPGAs, DSPs or GPUs can be used as coprocessors. New embedded GPUs that are capable of GPGPU (General-Purpose computing on Graphics Processing Units) with lower power consumptions seem to be an option for the future utilization on a Spacecraft, but currently out of bounds of the project [1]. The hardware components are mainly COTS. The main components will be complemented by:

- voltage regulators,
- voltage monitors,
- current limiters,
- latch-up detection circuits and
- a watch-dog timer.

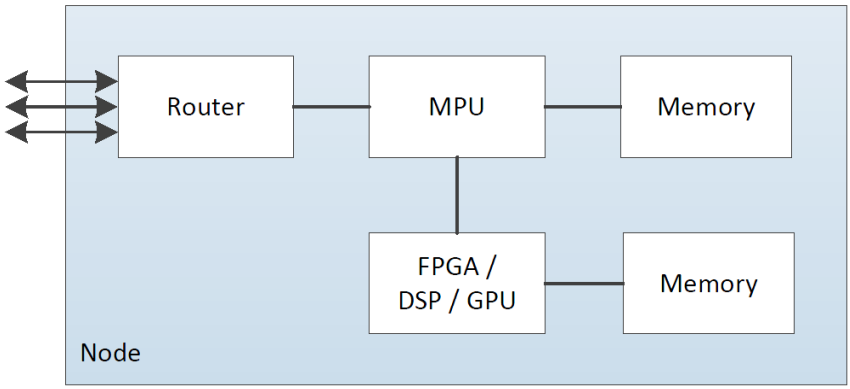

Figure 3-6: Processing Node of OBC-NG [1]

3.6.2. CPU Selection

For our test setup, we investigated the available CPU architectures, to find out which fits best to our requirements. We took a deeper look at four common

architectures available on the market: ARM, PowerPC, SPARC V8 and X86.

ARM

The ARM architecture has the highest market share in the embedded systems domain, resulting in a huge variety of microcontrollers, tools and software development kits. An ARM processor focuses on performance per watt and is often integrated in high–frequency microcontrollers nowadays. The widespread integration of ARM processors in smartphones and other embedded devices (e.g. Xilinx switched from PowerPC to ARM) reveals the general trust in ARM processors by the embedded industry.

PowerPC

The PowerPC architecture is well known for decades in the embedded world, being applied also in other domains, like servers. The PowerPC architecture focuses on high computing performance by moderate energy consumption. Only two vendors contribute to the further development of the architecture: IBM and Freescale. IBM concentrates its effort on server applications, whereas Freescale remains as the only provider for the embedded market. Over the last years, it was recognizable that the PowerPC architecture lost more and more of its importance in the embedded world.

X86

This architecture provides the highest single-threaded computing performance of all architectures, with the drawback of being very energy consuming. The plain high energy consumption is in severe contrast to the deficits of power on spacecraft. Nevertheless, positive is that this architecture enjoys the biggest support, regarding operating systems, programming languages, libraries and tools, compared to all other architectures. However, the amount of specific ICs of that architecture for the embedded sector is rather low.

SPARC V8

SPARC is a RISC architecture developed by SUN Microsystems. The SPARC V8

specification [15] was used by ESA to develop the LEON2 CPU. Later Gaisler continued the development and released the LEON3 and LEON4 processers yet. The LEON CPUs are widely used in the European space projects. LEON CPUs are available as soft-cores for FPGA designs or as microcontroller from Airbus D+S, Atmel and Aeroflex-Gaisler. Because of the older silicon-technology used for the ASICs, the maximum frequency is around 100 MHz.

Based on this evaluation on different architectures, the OBC-NG team decided to use ARM architecture. One of reasons was, that are the future perspectives and support by Xilinx. With the Zynq product group, Xilinx offers SoCs with an ARM CPU, standardized interfaces to peripherals and an FPGA. This facilitates the board-design of a node enormous. Additionally to that, Space Micro announced in September 2015 that they have delivered their CubeSat Space Processor (CSP) to NASA Goddard Space Flight Center. The CSP is radiation tolerant space processor, based on Xilinx Zynq 7000 SoC [11].

Atmel is an important space-qualified FPGA, ASIC, memory and microcontroller. Already today, the company offers a huge variety of ARM-based MCUs within their SMART product line. Some of them are adapted to meet low rad hard requirements, like the ARM32-SAM3X8 with an ARM Cortex-M3 processor. Atmel plans to increase their product portfolio on ARM-based aerospace processors by targeting more powerful architectures like the ARM Cortex-M7.

3.6.3. FPGA Selection

Based on the experience with Xilinx products of all project participants and the existence of powerful and space-qualified FPGAs within the portfolio of Xilinx, the decision was taken to use an FPGA product of this vendor.

3.6.4. Network Interface

SpaceWire has been selected as communication network technology, which is

widely in use within the aerospace sector, capable to establish a serial, bidirectional, full-duplex point to point connection with a maximum transmission speed of 400 Mbit/s. To establish a connection between two links to transfer data, the transmitter also has to receive control characters, which indicate that the buffer on the receiver side is sufficiently freed. This buffer is needed to store the incoming message. In case there is not enough buffer freed, the control characters indicate this and an overflow is prevented.

A crossbar implementation allows a fully meshed network, with each router of each node interconnected with each other. In OBC-NG, the router resides in an dedicated FPGA. The router contains an arbiter, which is in charge of the further distribution of the packets, based on the port they arrive. To reduce the latency of a message transmission, wormhole routing is used.

Targeting the real-time capabilities of the system, Spacewire is able to distribute time, which is of need to synchronize the real-time clocks of all nodes where necessary. This is conducted by an integrated immediate time code transmission. The still remaining latency is depending on the amount of interconnected links but predictable towards a maximum value [1].

4. Example Applications

Within this Chapter, the applications that have been ported to the OBC-NG system are introduced.

4.1. ATON

The ATON project studies an autonomous soft landing scenario for the moon. ATON (Autonomous Terrain-based Optical Navigation) focuses on the navigation part of such a mission, which is highly depending on optical navigation algorithms. The ATON software framework was already tested on common commercial computers but running the software on space-qualified hardware seems infeasible.

The software framework of ATON is separated into multiple modules that run mostly independent from each other but they are synchronized with respect to the input, e. g. when a camera image is captured the respective modules are triggered. Because the navigation filter (also a module) collects the results of the modules, the outputs of each module are under specific real-time requirement. Figure 4-1 shows the data flow between the modules of the framework.

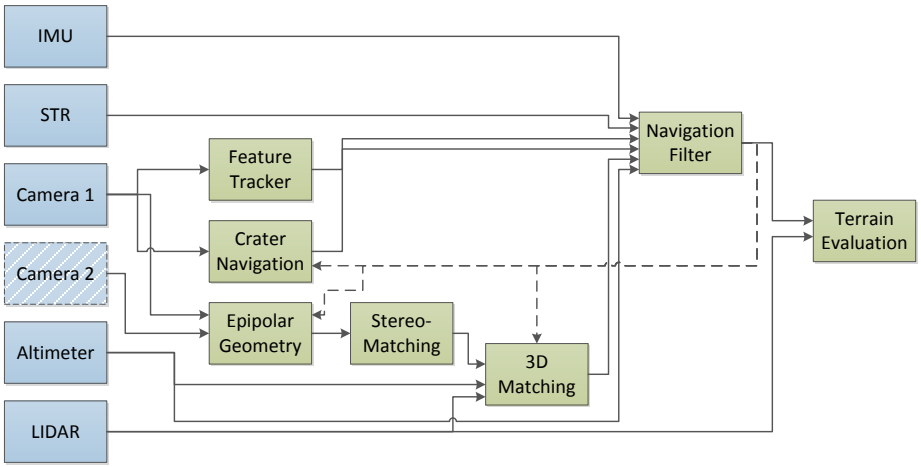

Figure 4-1: Block diagram of the data flow of the ATON software framework

The blue boxes are the sensors, and the green ones are the main software modules. The *feature tracker* observes the optical flow within consecutive images with respect to significant features. The *crater navigation* identifies craters on the moon surface and compares them to a catalog to estimate the absolute position of the spacecraft with respect to the craters found in the image.

Two camera images taken at different locations during the ascent are processed via the *epipolar geometry* module and the *stereo matching* module to acquire a 3-dimensional surface representation that is used to localize the spacecraft within a stored map. A second imaging technology (LIDAR) is used to directly acquire a 3-dimensional surface via range measurement for each sensor pixel. This is also used to match the images against a stored map. Lastly, the *terrain evaluation* analyzes the landing zone for hazardous objects and general constraints for a safe landing.

4.2. Cloud-Detection Application

Cloud detection is an important application on modern earth observation satellites. To save valuable download capacity useless image data with high cloud coverage should be excluded from a transmission. The detection of clouds in images is also often needed for further remote sensing applications. In figure 4-2 the typical input and output of the cloud detection algorithm is shown.

For the project OBC-NG the standard ACCA-cloud detection algorithm was implemented on top of the OBC-NG software framework, to test the system architecture, especially the processing power and network capability, with a real application. Experience should be gained in implementing an application in the system, particularly in the context of using the multi-node processing capability and the safety functionality of the OBC-NG system. Furthermore, the user friendliness of the implementation process should be examined. It should be evaluated, if there is any useful functionality that is worth being added or if there are some glitches that have to be fixed.

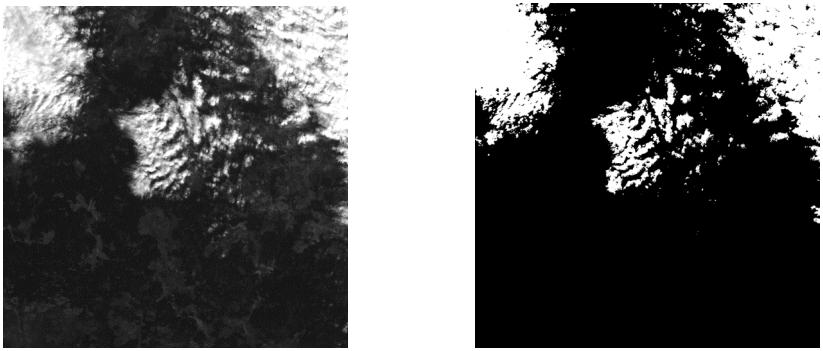

Figure 4-2 Left: cloudy Landsat satellite image; Right: cloud mask;

4.2.1. System design

The cloud detection application is unitized in three parts to fit the OBC-NG

multitask/node architecture:

- Sensor task (data input)
- ACCA task (cloud mask processing)
- Display task (displaying, saving resulting cloud mask)

Every task can be run on a different node. The OBC-NG middleware message protocol is used for the controlling and the communication of the tasks. More than one ACCA task can be used on different nodes to speed up processing performance. Below the schema of the demo application is shown, using three OBC-NG nodes for processing.

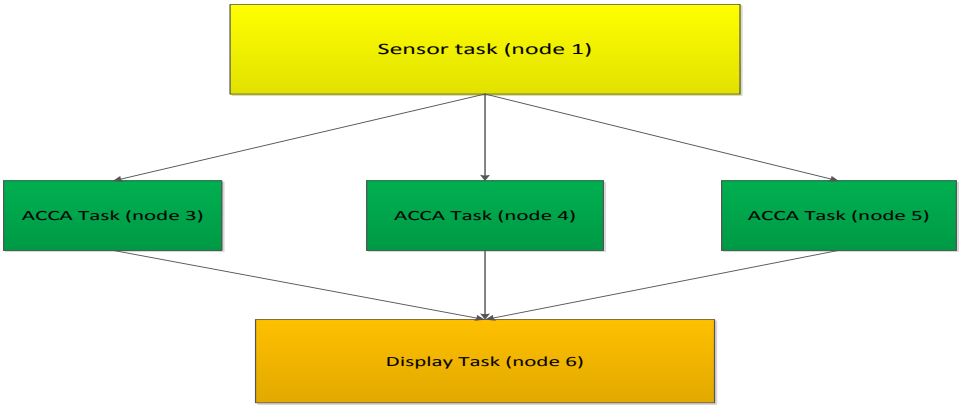

Figure 4-3: Schema of the ACCA cloud detection application on OBC-NG

Description of the tasks:

- **Sensor Task:** The sensor task simulates a sensor by reading image data from a database and sending the data to the ACCA tasks as a file. One message file consists of four 2048*2048 pixel sized 8-Bit grey value images (16 MByte), and the meta data, which is around 1 KB.
- **ACCA Task**: An ACCA task receives the big data message from the sensor task, extracts the data, processes the data, and sends the result

encoded in a message to the display task. For the processing step, an adapted version of the ACCA cloud detection algorithm [18] has been chosen using four channels (green, red, nir, swir) for creating the cloud mask.

- **Display Task:** When the display task receives an input message from one of the ACCA tasks, the result image is extracted and displayed or saved on the node, where the display task is running.

The core application behind the sensor, ACCA and Display tasks has been implemented as C++ classes on the Linux version of the OBC-NG runtime system. These classes are independent of the OBC-NG software framework. The OpenCV [12] and boost libraries [13] have been utilized. Nevertheless new (interface) class member functions had to be created to cope with the OBC-NG message protocol. After the classes have been extended to fit to the OBC-NG protocol, the implementation in the framework is relative straightforward using the OBC-NG middleware message API.

4.2.2. Conclusion

An implementation of the ACCA cloud detection algorithm could be integrated in the OBC-NG runtime system and tested on the OBC-NG prototype system successfully. The application core code is strictly written in the C++11 standard, with heavy usage of the C++11 standard data container classes and member functions. It was successfully built and executed on the OBC-NG prototype system, after updating the building environment (compiler, libraries). The OpenCV and Boost libraries could also be successfully built for the OBC-NG prototype system from source. Therefore, using a modern software development environment is possible to create applications for the OBC-NG Linux runtime system.

5. Implementation Status

This Chapter targets the current implementation status. All software and hardware implementations are listed below to provide an overview of all tasks that have been conducted.

5.1. Software Implementations

The software implementation is subdivided in layers, as they are described in the software architecture. The structure of the subsequent subsections follows that architecture in a top-down approach, from application, over middleware, down to operating system layer.

5.1.1. Application Development Toolchain

To facilitate the development of applications, a toolchain framework has been integrated: one for PetaLinux and another for RODOS.

5.1.2. Applications

On the application level, three major applications have been ported to OBC-NG. Two of them have been explained in this document. All three are: the ATON project, the Cloud detection application and the Far Range Navigation project.

The **ATON project** is able to run on the nodes of the OBC-NG, with PetaLinux as OS. The performance evaluation of the integration can be found in Chapter 4.1.

Additionally, the **Cloud detection Application** is ported to the OBC-NG, also running on PetaLinux nodes. An explanation of this application can be found in

Chapter 4.2.

In addition, several applications have been developed to demonstrate different aspects of the system software stack of OBC-NG. For instance, a setup to demonstrate the failure and reconfiguration for an ATON-like scenario is shown in Figure 5-1. The demo shows that the failure of an interface node with a navigation camera attached, leads to an automatic reconfiguration to the secondary camera. Additional failures of nodes show the capability to survive the loss of the master node. Nodes can be disabled until the last remaining node enters the safe mode.

Figure 5-1 OBC-NG navigation camera setup

5.1.3. Middleware

All three subsequent following layers of the middleware have been integrated in both operating systems (RODOS and PetaLinux). As basis of the layering model, the high-level network layer (OBC-NG Network Protocol) has been installed. On

top of that, the Tasking Framework is used, concluded with the Management layer. A special emphasis has to be set on the Reconfiguration module within the Management layer. To access the Tasking Framework and the Management layer an API is provided, necessary for the application developers to access the system.

5.1.4. Operating Systems

As non-real-time operating system, PetaLinux has been integrated. It is integrated as multi-core operating system. RODOS comes with multi-core support by a shared memory system to enable communication between cores within a single node. The RODOS is the real-time operating system for OBC-NG.

5.2. Hardware Implementations

5.2.1. Zynq breadboard

For the evaluation of the OBC-NG system, we developed and made a breadboard. The OBC-NG breadboard consists of a baseboard, which holds three mini-modules with a Zynq-System. We bought the mini-modules of the type TE0720 from the company Trenz. Every Zynq module has the following features [14]:

- Xilinx XC7Z020 system-on-chip with programmable logic
- 1 Gbyte DDR3-SDRAM
- Gigabit Ethernet transceiver
- 32 Mbyte QSPI flash memory
- 4 Gbyte e-NAND flash memory
- power converter for core voltages
- system controller CPLD

The Xilinx Zynq is divided into two parts:

- The processing system (PS), consisting of the two ARM CPU cores and several peripherals like memory controller, Ethernet interface, GPIO, etc.
- The programmable logic (PL), which is connected to the PS by a central interconnect module.

The base board provides the following features:

- Ethernet connector for every node,
- COM to USB converter of the type Cypress CY7C64225 and a mini-USB socket for every node,
- LVDS driver and two mini-Sub-D sockets per node for SpaceWire,
- two interlink connections per node providing 11 differential signal lines to the two other nodes,
- 4 Mbyte synchronous SRAM for the PL,
- 10 LEDs - 2 on PS and 8 on PL,
- 8 pin IO strip connected to the PL,
- 1 button connected to the PS,
- CAN transceiver for every node connected to the CAN-bus on the board,
- Current and voltage monitor IC (Texas Instruments INA230) connected to a common I²C-bus.

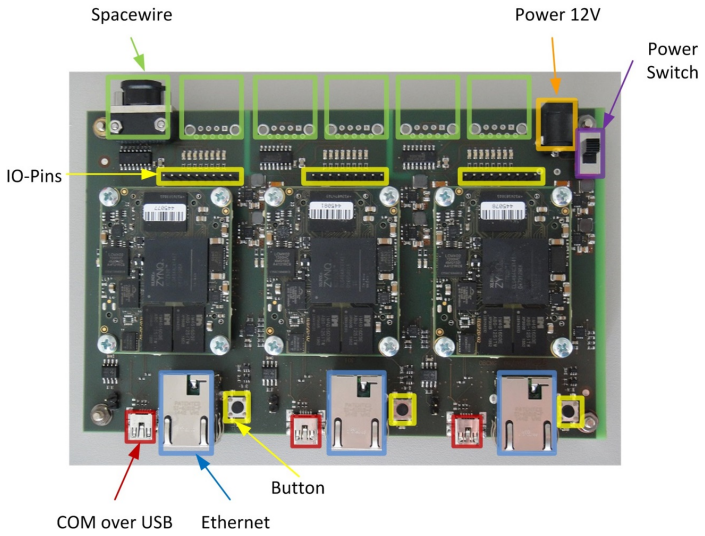

Figure 5-2 Top view of the OBC-NG bread-board

5.2.2. SpaceWire router

Because SpaceWire was chosen as the network technology a router was implemented by use of VHDL. Main attributes of the router are**:**

- Two types of interfaces are selectable. At one hand SpaceWire with its Data / Strobe signals. On the other hand a FIFO interface. The FIFO interface is useful in case the router shall be connected directly inside the SoC as it is planned for the AXI Interface.
- Non-blocking crossbar to enable arbitrary connections between inputs and outputs.
- Priority of addresses. The priority is applied during arbitration. This means in case multiple inputs are trying to get access to the same output, the input with a prioritized address will get precedence.
- Maximum packet size. The maximum packet size can be determined to prevent faulty nodes from blocking data paths for an indefinite time. In

case the allowed byte amount is exceeded, the packet is closed by EEP automatically and the rest of the packet inside the input buffer is removed.

- Transfer timeout. The timeout is another mechanism to prevent data paths are blocked for an indefinite time. This blocking would happen in case a packet transfer was started but stopped before EOP. In case the packet was not transferred in a predefined amount of time, the error handling will be the same as for the maximum packet size violation (EOP plus packet removal).
- Timecode distribution. Timecodes initiated at the inputs are immediately transferred to all outputs. But with the constraint that the actual timecode value is incremented by 1 compared to the previous timecode transfer. Without this constraint it would be possible to create loops inside the network.
- Configuration by RMAP protocol (ECSS-E-ST-50-52C). Configuration is accessible by every input port.

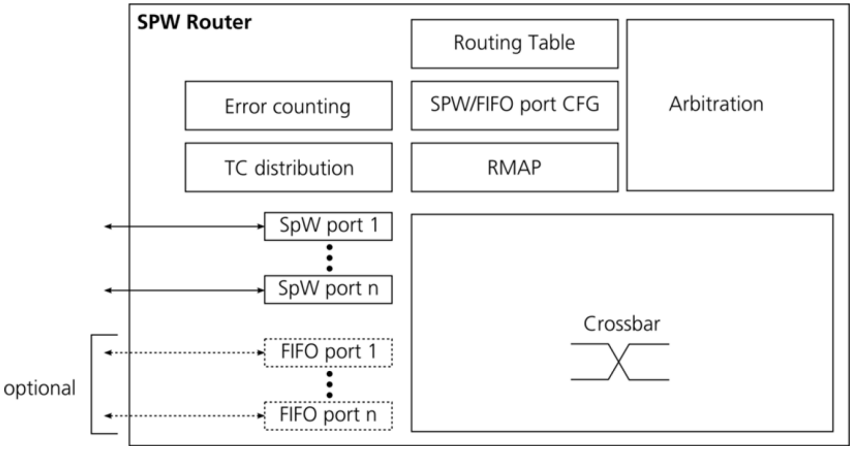

Figure 5-3 Block diagram of SpaceWire router

6. Remaining Issues

Systems, including the OBC-NG project, are always a careful combination of holistic and specific approaches. Some parts of the system might be very generically designed to be able to target general purposes and other parts are specifically designed to fulfill a special scenario in an optimal fashion. Such a combination can lead to unexpected bottlenecks and shortcomings, which become visible when the project is actually carried out. A reason for this is that the gained knowledge, by running the project, might has changed the pre-project perception of requirements and conditions.

Within the following subsections these discovered bottlenecks and shortcomings are denoted, offering solutions to overcome the same. In focus is the design, the implementation, the application as well as the evaluation.

6.1. Hardware Design Issues

Regarding the SpaceWire router, the integration into the node is still missing. The remaining tasks are:

- The router must be connected to the internal AXI bus of the Zynq. An IP core with the already existing AXI interface must be generated and verified.
- A Linux driver must be developed. Maybe the router interface to the software hast to be changed, if problems with driver data handling occur.
- The performance of router and driver must be evaluated.

6.2. Software Design Issues

The OBC-NG Network Protocol waits for a specific time to be sure, that the

system has finished the reconfiguration. The section in the document states:

"After a certain timeout, which has to be configured mission-specifically, the node will delete all of its pending messages, switch into the new state and go back into the normal running state. It is recommended to wait again a short time before sending new messages so that the probability that all nodes in the network are now synchronized to the new state is sufficient."

The shorter the waiting time, the higher is the threat of not receiving all "finished reconfiguration" messages of all nodes. Moreover, the longer the waiting time, the higher the threat that the 5 seconds reconfiguration time goal is out of bounds. Besides, only the Master is informed about which nodes have finished the reconfiguration successfully.

New mechanism should be developed to reduce the preparation time for reconfiguration and the waiting time after reconfiguration for all nodes. The monitoring service should be refined. The characteristics monitored should be chosen. The suitable frequencies of monitoring including observation should be figured out.

At the beginning of OBC-NG we expected, that we have to change the OS significantly, especially in the field of scheduling. Having a deep understanding of the OS and a direct access to the source code was very important to us. That is why we choose RODOS as real-time OS. It turned out, that we could handle everything in the middleware. Not touching the OS, make us more independent regarding OS and hardware-platform. The shift of other DLR projects away from RODOS towards RTEMS (Real-Time Operating System for Multiprocessor Systems) leads to the conclusion, that follow-up projects should also consider the switch to RTEMS.

We addressed SEU issues sparsely in the OBC-NG project yet. Because we cannot

rely on perfectly working hardware, the software must compensate most of the SEU errors. In the future, we must define and implement monitoring and mitigation techniques on several levels of the software stack. The FPGA must be protected against SEU errors as well.

6.3. Implementation Issues

For application programmers, the system configuration settings are not easy to understand and manage. Therefore, a Graphical User Interface (GUI) to give an overview of the components of the system would be supportive. This model of the system could be used to facilitate the configuration of the entire system, for example by enabling the mapping of application tasks to specific nodes.

Also of interest would be a concrete integration of a network simulation tool within the development process, to evaluate the network protocol implementation. Network simulation tools allow a sealed investigation of the network protocol implementation, to verify its correctness at an early stage of the development. OMNett++ (https://omnetpp.org/) had been used for that purpose, but it was just loosely connected to the development process. A tight bootstrapping of such a tool would be beneficial. Additionally, this setup could be used to define test-cases for the later evaluation of the entire communication network.

6.4. Application Issues

The API of the middleware needs to be further improved to accelerate and facilitate the application development. In addition to that, it would be helpful to have faster access to a running system. This could be staged, with initial access to the software and subsequent access to the entire system. Jenkins (http://jenkins-ci.org/) can be used to allow continuous integration. At first stage, software tests could be carried out on a remote Jenkins server. At second stage, the actual hardware could be attached to the remote Jenkins server, enabling

hardware in the loop testing.

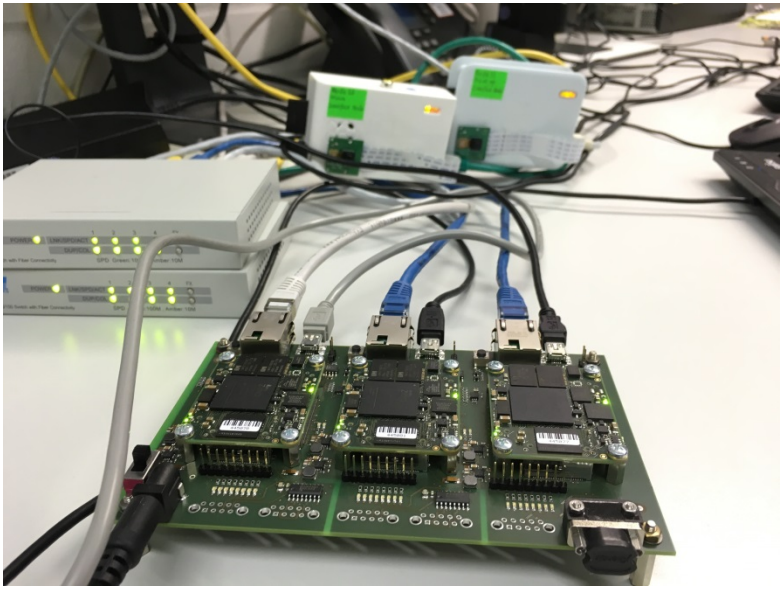

Figure 6-1 OBC-NG bread board and two interface nodes.

6.5. Evaluation Issues

Reliability is an important aspect in space missions, and it is even more in focus when a new concept with a different approach to reach the required reliability is used for the first time. The OBC-NG project, with its dynamic reconfiguration concept, is in such a position. Therefore, a validation of the reliability of the entire system should be considered. An appropriate mean for this would be the creation of a Timed Failure Propagation Graph (TFPG) on system level. For the OBC-NG project, a TFPG could be generated of a loosely timed system model with a linked reconfiguration model and potential failure model (Failure Mode and Effects Analysis Table). A model checker can then process this TFPG. In case the probabilities are also desired, the expected likelihood of a failure has to be

integrated in the linked failure model.

7. Conclusions

This document gave an overview of the goals and the achievements of the DLR project OBC-NG. It was established to develop an on-board computer architecture for future space missions with high demands for computational power. After an initial study phase of one year, a three-year project was carried out to reach the Technology Readiness Level 4 (component and/or breadboard functional verification in laboratory environment). The project goals will be reached at the end of this, as this document has shown. A breadboard computer, based on COTS components, was designed and manufactured and a software stack developed to implement the envisioned system architecture. In parallel, several studies to investigate the state-of-the-art in relevant areas and representative applications have been developed and ported to the target system. These applications show, which kind of missions could benefit from OBC-NG-based computing architecture.

Besides the breadboard demonstrator, the main contribution of this project is the scalable architecture of a reconfigurable, distributed on-board computer system. This architecture is independent of the underlying hardware and operating system. The developed middleware can be easily ported to other processor architectures and operating systems. It is possible to combine different types of processors, FPGA and other coprocessors to build a flexible, heterogeneous on-board computer, which can be tailored for a specific mission. Even if the selected COTS Zynq platform performs poorly under space conditions, the established software and FPGA implementations can be ported to different hardware with reasonable effort.

We think that the results of this project are promising, considering the available resources. They build a good foundation for further developments. The hardware is not yet flight-ready; however, the current design is a good start to

build a flight model in the future. The software stack is quite complex, it spans from the low level operating system over a new middleware to the high level application programming interface. All this areas need further improvements and some refactoring, especially in the area of error detection and recovery below the node level, and a more user-friendly API. However, no fundamental reimplementation seems to be necessary.

Besides the mentioned improvements to the system, we want to combine the OBC-NG high performance COTS nodes with "classical" radiation-tolerant LEON3 processor nodes in a follow-up project. These nodes will be integrated by the OBC-NG middleware to the OBC-NG network and could act as master nodes to improve the reliability of the overall system. It would then be possible to build a cluster of radiation-tolerant processors for deep space missions, where the COTS components may not suitable to withstand the increased radiation.
We are confident that the OBC-NG architecture provides the means to support a large variety of future space missions with the necessary computational power.

List of Figures

List of Tables

References

[1] D. Lüdtke, K. Westerdorff, K. Stohlmann, A. Börner, O. Maibaum, T. Peng, B. Weps, G. Fey, A. Gerndt: "OBC-NG Towards a Reconfigurable On-board Computing Architecture for Spacecraft," in *Aerospace Conference, 2014 IEEE*, Big Sky, Montana, 2014.

[2] ESA-ESTEC Data Systems Division / Microelectronics Section, "Techniques for Radiation Effects Mitigation in ASICs and FPGAs," Noordwijk, The Netherlands: ESA-ESTEC, 2012.

[3] S. Asano, T. Maruyama, Y. Yamaguchi, "Performance Comparison of FPGA, GPU and CPU in Image Processing," in *FPL*, Prague, 2009.

[4] D. H. Jones, A. Powell, C. Bouganis, P.Y.K. Cheung, "GPU versus FPGA for high productivity computing," in *International Conference on Field Programmable Logic and Applications*, Milano, 2010.

[5] I. V. McLoughlin, Timo Bretschneider, "Achieving Low-cost High-reliability Computation Through Redundant Parallel Processing," in *ICOCI*, Kuala Lumpur, 2006.

[6] P. Murray, T. Randolph, D. Van Buren, D. Anderson, I. Troxel, "High Performance, High Volume Reconfigurable Processor Architecture," in *IEEE Aerospace Conference*, Big Sky, 2012.

[7] J. Samson, G. Gardner, D. Lupia, M. Patel, P. Davis, V. Aggarwal, A. George, Z. Kalbarcyzk, R. Some, "High performance dependable multiprocessor II," in *IEEE Aerospace Conference*, Big Sky, 2007.

[8] J. Samson, E. Grobelny, S. Driesse-Bunn, M. Clark, S. Van Portfliet, "New Millenium Program Space Technology 8 Dependable Multiprocessor: Technology and Technology Validation," *Journal of Spacecraft and Rockets,* 2012.

[9] R. Mikkilineni, I. Seyler, "Parallax - A New Operating System Prototype Demonstrating Service Scaling and Service Self-Repair in Multi-core Servers," in *20th IEEE International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises*, Paris, 2011.

[10] R. Mikkilineni, "Parallax - A New Operating System for Scalable, Distributed and Parallel Computing," 2011. [Online]. Available: https://software.intel.com/en-us/articles/parallax-a-new-operating-system-for-scalable-distributed-and-parallel-computing. [accessed 22-11-2015].

[11] S. Micro, „Radiation Tolerant CHREC Space Processor," Space Micro, 23 4 2015. [Online]. Available: http://www.spacemicro.com/assets/datasheets/digital/slices/CHREC.pdf. [accessed 19-11-2015].

[12] „OpenCV Homepage," [Online]. Available: http://opencv.org.

[13] „Boost C++ Library," [Online]. Available: http://www.boost.org.

[14] A. Lukats, „TE0720 GigaZee - TE0720 GigaZee Zynq SoM - Trenz Electronic Wiki," 22 9 2014. [Online]. Available: https://wiki.trenz-electronic.de/display/TE0720/TE0720+GigaZee. [accessed 22-11-2015].

2016-03

ISRN DLR-FB—2016-03