BOSS: Software and FPGA Middleware for the "flying laptop" Microsatellite

Sergio Montenegro*, Hans-Peter Röser**, Felix Huber***
Presented by Sergio Montenegro.

*FhG FIRST
Kekulestr 7
12489 Berlin
www.first.fhg.de/~sergio
sergio@first.fhg.de
Tel +49 30 63981878

**Universität Stuttgart
Institut für Raumfahrtsysteme
Pfaffenwaldring 31
70550 Stuttgart
roeser@irs-uni-stuttgart.de

***Steinbeis Transferzentrum Raumfahrt
Rötestr. 15
71126 Gäufelden
huber@tz-raumfahrt.de

Abstract

The real time operating system BOSS was design for dependability and to support fault tolerance by its integrated middleware. Now we are implementing its middleware in FPGA hardware, to allow a simple communication between software-software, software-hardware, hardware-hardware and hardware-software modules without having to differentiate if the communication partner is a software or a hardware module. This novel communication principle will be a benefit to the flying laptop, because its control system is mainly implemented in FPGA-Modules.

1. Introduction

BOSS (an FhG FIRST development) is a real-time embedded operating system with integrated middleware, designed for safety and simplicity and to conduct its own mathematically formal verification. Complexity is the cause of most development errors – by eliminating complexity, most development errors can be eliminated. This was one objective of BOSS. The BOSS middleware's simplicity allows the system to be easily understood, used and ported to other platforms, even to FPGA (Field Programmable Gate Array) logic.

As part of the Stuttgart Small Satellite Program, the University of Stuttgart's Institute of Space Systems is developing and building several small satellites. The "Flying Laptop" (Figure 1) is a microsatellite with a mass of less than 100 kg. Power is delivered by three solar panels, two of which are deployable. The satellite is three-axis, stabilized by momentum wheels, which are desaturated by magnetic torquers. The two main objectives of the mission are Earth observation and technology demonstration: VIS/NIR camera, TIR camera, VHF, UHF, S-band, Ka-band, FPGA based on-board computer.

The FPGA on-board computer allows the satellite to operate in a so-called "rent-a-sat" mode. The satellite will be offered to interested companies as a software testing platform to verify their algorithms or firmware under real space conditions. In the case of an emergency, the reconfigurable computer architecture enables full control of the satellite to be regained within milliseconds by means of a simple reset.

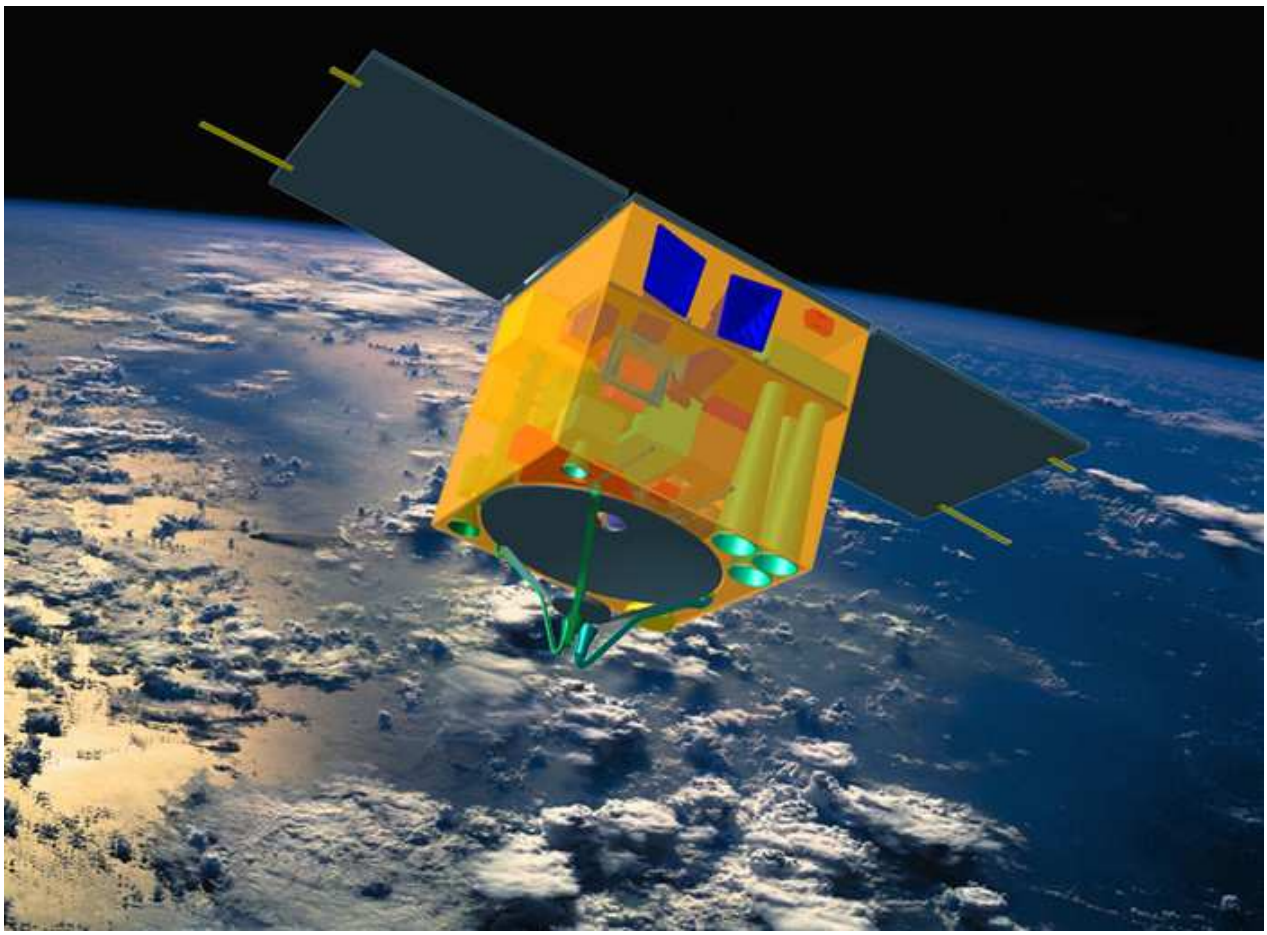


Figure 1: The "Flying Laptop" satellite

2. The “Flying Laptop”’s Control System

The "Flying Laptop"’s on-board computer differs significantly from other on-board computers: In the nominal mode, the OBC will operate solely with algorithms implemented directly in hardware. No control Software will be running. This is nowadays possible by the Celoxica Handel-C compiler tool that allows for the direct generation of an FPGA configuration by using a special version of the “C” programming language without the need for a HDL or schematics. This method provides short development times and a complexity of the algorithms that is otherwise hardly to achieve. The direct hardware synthesis provides the fasted processing speed and has a hard real time behaviour without any latencies.

In the rental mode, a more classical approach can be provided to customers using a combination of traditional software programs and hardware FPGA algorithms. Nowadays, the border between hardware and software is disappearing, at least inside FPGAs. The currently available larger FPGA chips include a CPU in the same chip together with the FPGA functionality (figure 2). This allows us to combine traditional software programs (CPU-Memory programs) with hardware algorithms. If we exploit this attribute, we can reach a real HW-SW Co-design and get a higher dependability and safety of the system specially as regards to time response. The remainder of this paper will describe the OBC concept when operated in the rental mode.

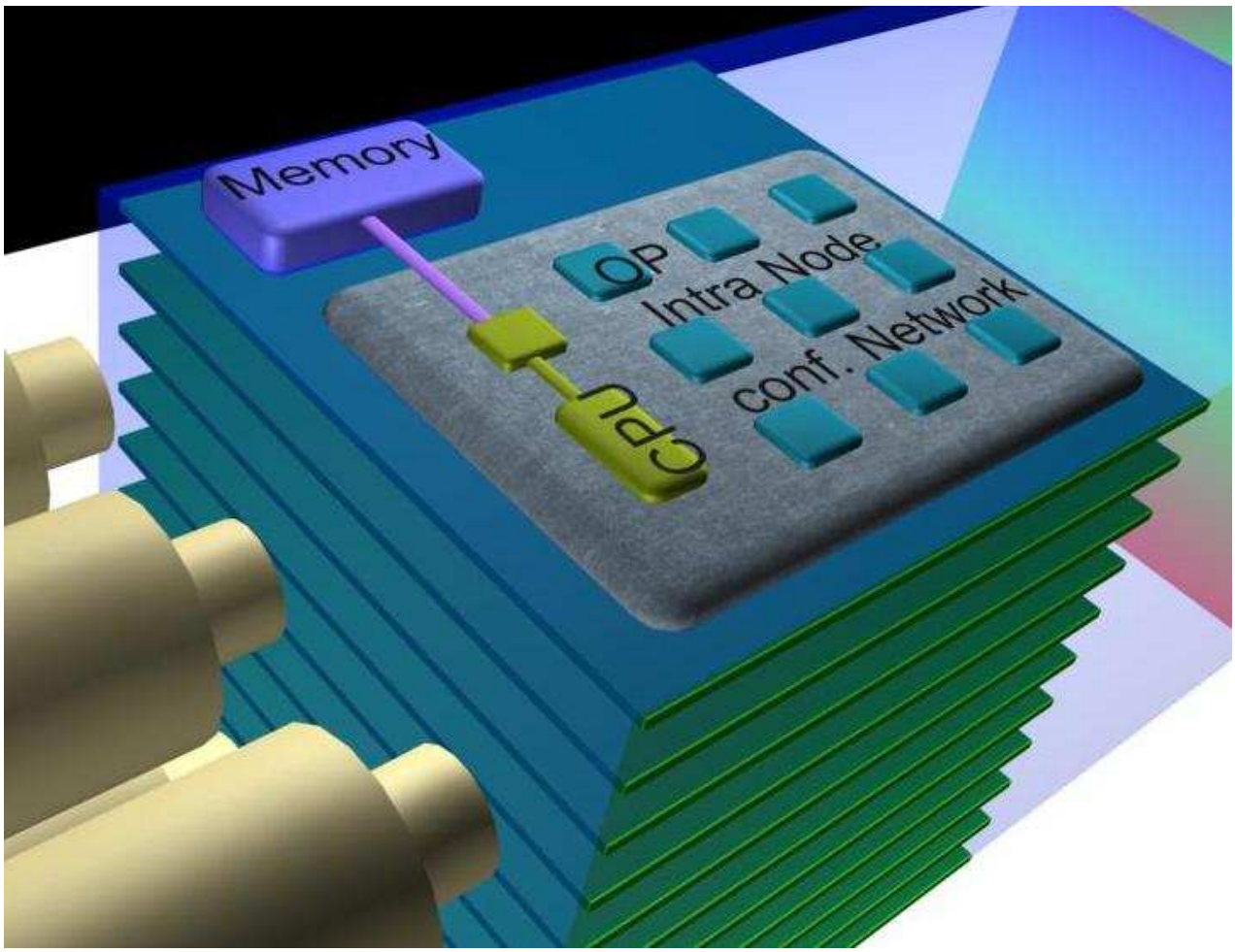


Figure 2: Example of an array of FPGAs with integrated CPU

To implement fault tolerance, the OBC will consist of up to four identical nodes, each of which with the same set-up. Three nodes will be available to the customer in rental mode while the fourth will act as a background sentinel that will reset the OBC only in case the customer software would bring the satellite into a harmful situation. The customer can choose whether to use the built-in microprocessor core or can have a soft-core loaded that is emulating the work-horse microprocessor of the customer, thus removing the need to reprogram some processor dependant functions. Some of the low level functionality such as the DRAM controller, SEC-DED for the memory, communication links between the nodes and I/O interfaces will still be provided by FPGA functionality and will be transparent to the customer.

The FPGA will have a scrubber process that continuously checks the configuration of the device for changes due to radiation. The scrubber itself is TMR protected using a fail safe external H/W voting and will reload parts or the entire FPGA configuration if an error was detected. The customer can then decide whether to implement a complete operating system and user software on its own with fault tolerance mechanisms or use the BOSS middleware as an operating platform that is described later in this paper.

FPGA algorithms represent a quantum step in terms of real-time dependability, as shown in Figure 3:

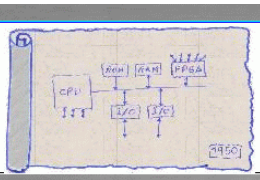
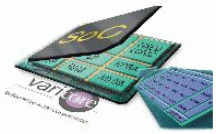
	CPU-Mem. Programs 	FPGA Algorithms 
1	Loadable	Loadable
2	Passive	Active
3	Sequential	Parallel (maximal)
4	Interrupts: another paradigm	Dedicated hardware
5	Reaction time has a big jitter (max.. min)	Reaction time jitter < HW clock time
6	Time challenging deterministic	Time deterministic
7	Resources sharing by time	Dedicated hardware
8	Building blocks alter the time behaviour	Building blocks do not alter the time behaviour
9	Difficult to build by composition	Simple to build by composition

Figure 3: Comparison of CPU-memory programs and FPGA algorithms

3. The BOSS Middleware Principle

In the rental mode, we allow users to load their own programs to control the satellit, but the low level functionality to access the satellite devices is implemented as a hardware function. To enable us to merge both worlds, we need a communication mechanism that allows applications to communicate, regardless of whether they are implemented in software or hardware. This is achieved by means of the BOSS middleware. The BOSS middleware has already been implemented and is being used in software for real-time dependable systems. We now aim to implement the same middleware in FPGA hardware. This will bring hardware and software developers together in an environment that is familiar to both. It makes no difference whether applications running on top of the BOSS middleware are implemented in software or hardware. For communication purposes, it makes no difference whether the communication partner is implemented in hardware, software or both. The BOSS middleware allows any combination of communication (SW/SW, SW/HW, HW/SW, HW/HW) and creates a standard interface between software and hardware without needing different device drivers for different implementations.

The results already achieved are very promising. BOSS has been in continuous use in space (BIRD satellite) and in medical devices and other control systems for a number of years now. Other satellite projects such as CubeSats and the "Flying Laptop" are now being developed. The highly modular operating-system software was implemented by using the latest software technology and the critical parts were formally verified. The applications running on top of BOSS are implemented using object-oriented technology, resulting in highly modular application software. To achieve a well-structured application system, we defined a software backplane, the BOSS middleware. The principle of a software backplane (middleware) allows easy configuration of the system by simply plugging the software components into and out of the SW backplane, as is done in hardware.

4. Description of the BOSS Middleware

The BOSS middleware provides very simple communication mechanisms for applications running on the top of it, regardless of whether they are implemented in software or (FPGA) hardware (now under development). It was designed to support fault tolerance. All processes running on top of the BOSS middleware can exchange messages asynchronously using a subscriber protocol: a

process or hardware device can subscribe to one or more message types by name. When a process or hardware device sends a message of a given type (name), each subscriber to this name receives a copy of the message. For communication purposes, the node and even the software/hardware barriers/boundaries are transparent. The messages are distributed across these barriers. Using this approach, very high flexibility is obtained and users do not have to distinguish between local/remote functions or hardware and software functionality. The system can be configured or reconfigured simply by plugging software modules or hardware devices into or out of the middleware.

The BOSS middleware provides transparent support for fault tolerance. The simplest example of this is a controller sending commands (messages) to a device. As a first step, we insert the middleware between the device and the controller by implementing the same interface on both sides of it. Neither the controller nor the device notices this intervention. The middleware forwards the messages across node boundaries, which means that controller and device no longer need to be located in the same node. Furthermore, messages can be replicated if there is more than one subscriber to a message type (name). Now we can add a monitor to hear messages of the same type that the device hears. The monitor can create a log file and/or execute an online diagnosis of the system. Again, no one will notice this intervention (see Figure 4).

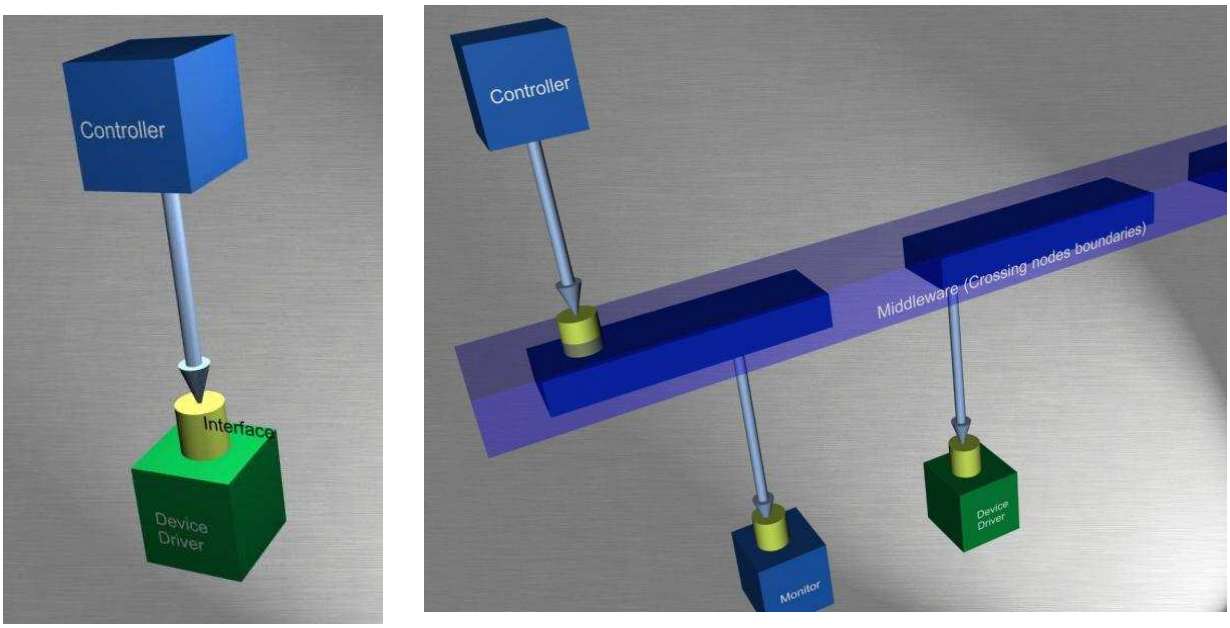


Figure 4: Middleware insertion

The next step is to replicate the controller, simply by creating several instances of it, if possible running on different nodes. They need not know about the existence of the other replicas. What are needed now are voters that intercept all messages to the device, compare them and send only those that are most likely to be right (a democratic decision, e.g. two of three) to the device. If required, it is possible to replicate the voter, too. One voter – the worker (as in BIRD) – is in charge and the other – the supervisor (as in BIRD) – is a hot redundancy. The supervisor is ready to take control if the voter in charge fails to respond (see Figure 5).

The routing of messages depends only on the types/names of the messages and on who is subscribed to each name.

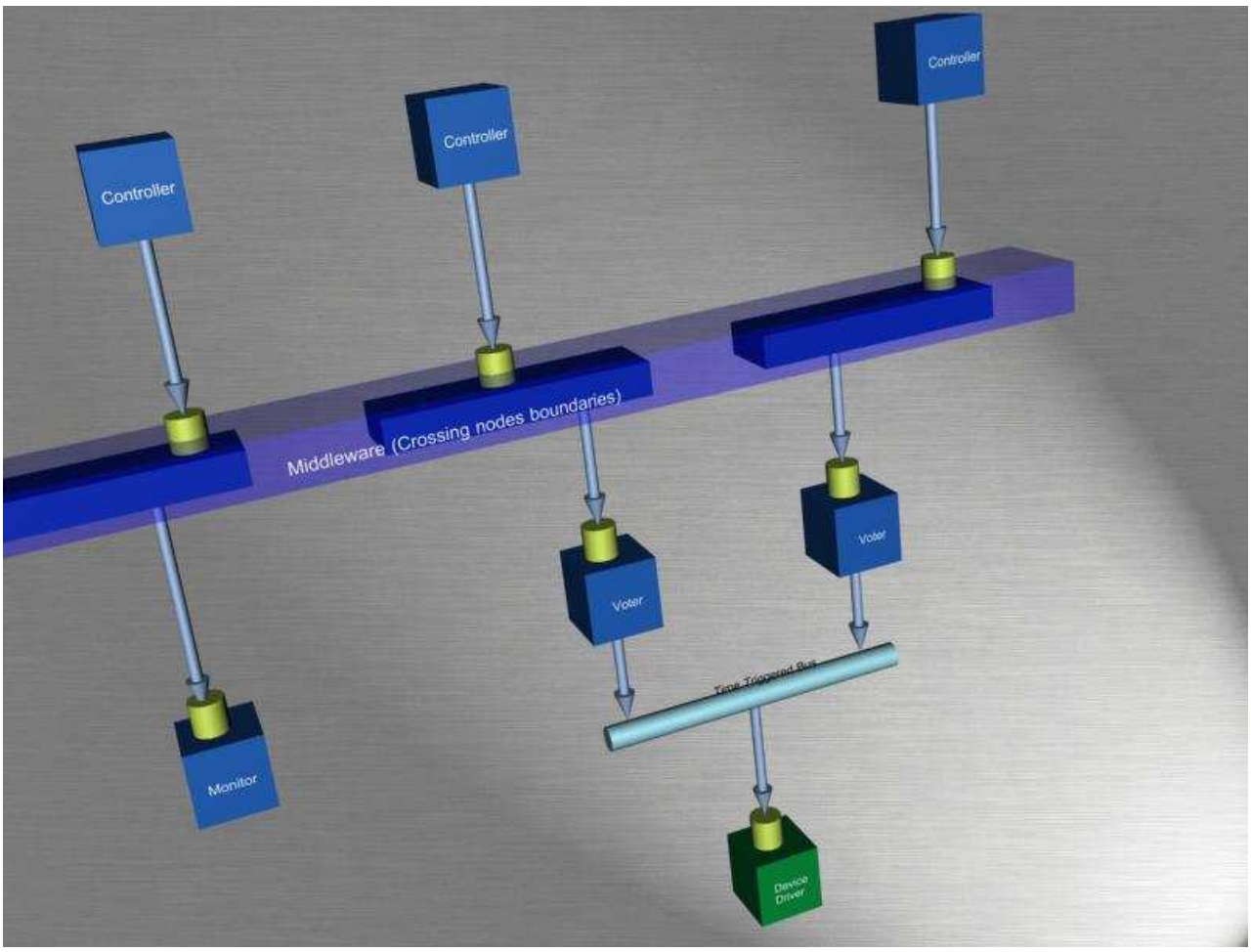


Figure 5: Middleware and voters

References

1999: Montenegro, S.: Entwicklung sicherheitsrelevanter Systeme, Hanser Verlag, ISBN: 3-446-21235-3

2003: Briess, K., Baerwald, W., Gill, E., Halle, W., Kayal, H., Montenbruck, O., Montenegro, S., Skrbek, W., Studemund, H., Terzibaschian, T., Venus, H.:

Technology Demonstration by the BIRD Mission

4th IAA Symposium on Small Satellites for Earth Observation, April 7-11, 2003

ISBN 3-89685-569-7

2002: Briess, K., Bärwald, W., Hartmann, M., Kayal, F., Krug, H.3, Lorenz, E., Lura, F., Maibaum, O., Montenegro, S., Oertel, D., Röser, H.P., Schlotzhauer, G., Schwarz, J., Studemund, H., Turner, P., Zhukov, B.:

Orbit Experience and First Results of the BIRD Mission

53rd International Astronautical Congress The World Space Congress 2002, October 10-19, 2002, Houston, Texas, USA

2002: Briess, K., Montenegro, S., Bärwald, W., Halle, W., Kayal, H., Lorenz, E., Skrbek, W., Studemund, H., Terzibaschian, T., Walter, I.:

Demonstration of Small Satellite Technologies by the BIRD Mission

16th Annual AIAAA/USU Conference on Small Satellites, Logan, Utah, USA 2002