Adaptive Hardware In Autonomous And Evolvable Embedded Systems

Stephan Kubisch, Ronald Hecht, Dirk Timmermann
University of Rostock
Institute of Applied Microelectronics and Computer Engineering
18051, Rostock, Germany
{stephan.kubisch, ronald.hecht, dirk.timmermann}@uni-rostock.de

Abstract: *Meeting the high demands of outer space missions requires immense efforts and is exceedingly time consuming. For the latest accomplished missions, the engineering of the hardware and computing systems took years and must be completed before launch.*

The architecture of a Network on Chip (NoC) based System on Chip (SoCs) is presented in this paper. These SoCs capable of partial reconfiguration at runtime are a new paradigm in the embedded world and show advantageous implications to different problems. Autonomous systems can be designed which dynamically adapt to the environment within their functional scope, subsequent updates can easily be submitted as plain source code due to the integrated design flow, and the robustness is increased using an on-board simulation model.

FPGAs are used in space for a long time. In future space missions and other industrial domains, reconfigurable embedded SoCs as specified in this paper will play an increasing major role. Due to the growing functional spectrum, flexible and powerful architectures are needed. On the one hand, to easy and shorten the development process. On the other hand, to meet the high and stringent demands in the field.

Keywords: System-on-Chip (SoC), Network-on-Chip (NoC), Reconfiguration, Autonomy

1 Introduction

Outer space missions have been, are, and will be the most challenging adventures scientists are faced with. Whether they are manned or unmanned, these missions rely on an enormous extent on a faultlessly functioning system of computers, analogue sensors, and mechanical and integrated digital devices. These systems are designed to work in harsh and unknown territories. There, they will be confronted with different conditions compared to the clean rooms they are developed in. Examples are with different types of radiation, extreme temperatures, and less or no gravity as pointed out in [17], [19], and [23].

Xilinx [21] radiation hardened and Alteras [1] high reliability FPGAs are designed to cope with some of these hard environmental conditions. Another aspect is the long time it takes to plan, realise, and test hardware and software systems for space missions. Today, this takes years. For the latest accomplished missions to function properly, for example the Mars Exploration Rover Mission [18], various techniques have been used for the digital part. The main methods are Triple

Module Redundancy (TMR) [22] and complete reconfiguration of the FPGAs. TMR uses circuits in triplicate to benefit from majority decisions and thus reducing single-event upsets (SEUs) and transients (SETs). Unfortunately, TMR requires additional silicon resources, wiring, consumes more energy, increases design complexity, and it requires the engineering to be completed before start. Reconfiguration of the entire device is necessary to readjust the circuit in upset conditions. This method implies a short period of complete inactivity during the reconfiguration process, but not all applications allow interruptions. Further, updates or bug-fixes for in-flight reprogramming can only be submitted via an error prone wireless medium [11].

To shorten the time to the mission's start and to ease the whole development process, a new paradigm can be used in future projects. This paper presents an architecture for embedded systems using reconfigurable hardware based on a NoC approach. It relies on the FPGAs' ability of partial reconfiguration and integrates the design flow on the device, from the hardware description language (HDL) design entry to the final partial bitfiles. The design flow runs under a Linux operating system on the FPGA itself, which is adapted to the embedded PowerPC and modified for partial and dynamical reconfiguration at runtime.

The implications of this architecture are immense. The development time is shortened and designs can be updated after the spacecraft has launched. The system reveals self-x attributes like self-(re)configuration, self-optimisation, and self-healing as introduced by the research fields organic and autonomic computing. Due to the integrated design flow and its autonomy, the system can partially reconfigure and heal itself in critical situations. The system can organise and optimise itself over time. It adapts to the environment it is faced with and which is not really known before arrival as mentioned at the beginning. This increases the lifetime of such systems and, therefore, benefits the success of costly outer space programs [13].

The remaining part of the paper is sectioned as follows. In Section 2, the current state-of-the-art in reconfigurable computing is addressed. Section 3 describes the architecture of our NoC based SoC and highlights the integrated design flow. In Section 4, the expected advantages of our system architecture are pointed out. Finally, Section 5 describes the newsworthy progress of our work, emphasises some potential areas of application besides missions to space, and concludes the paper.

2 State of the Art in Reconfigurable Computing

From the Xilinx XC6200 series to the latest Virtex 4 generation of highly integrated platform FPGAs, it is possible to partially reconfigure an FPGA at runtime. Complete reconfiguration inevitable leads to temporary halts of the particular application. Many applications do *not* allow interruptions. The ability of partial reconfiguration opens up much more applications areas for reconfigurable devices. The first partial reconfigurable devices only supported column based reconfiguration. This technique was only hardly practicable. Wires crossing the areas to be reconfigured must not have been touched. Thus, engineers often refrained from using column based partial reconfiguration and relied on complete reconfiguration as so-called "'scrubbing"' technique [18]. However, the Virtex 4 FPGAs allow a tile based partial reconfiguration. Now, wires can be routed around the tiles easing partial reconfiguration. But even today, the main problem with partial reconfiguration still remains – the deficient support and availability of design tools. Two examples of third party tools for the relocation of tiles are presented in [9] and [12].

To manage the diversity of resources, hardmacros, and IP cores parallel operating on current platform FPGAs, various projects cope with operating systems (OS) for dynamically reconfigurable systems. Next to the general responsibilities of OS, the new tasks comprise dynamic reconfiguration, communication, interruption and relocation of the reconfigurable modules. Developments range from specialised hardware OS [20], on-chip reconfiguration managers [10] to extensions of a real-time Linux [2]. Most of these projects favour NoCs as communication medium. Formerly utilised bus system can not meet the high demands on bandwidth, speed and scalability any more.

Due to these trends, FPGAs are not any longer just one part of an embedded system but they represent the system itself. The latest available development boards are rather equipped like a personal computer or workstation than a single chip solution with some discrete components surrounding it. Upon this matter of fact, the following section describes our architecture of a NoC based SoC.

3 Network-on-Chip based Architecture

During the last years, a new paradigm has developed. In various projects, on-chip bus systems have been replaced by NoC architectures providing a homogeneous and powerful communication backbone between the components of the embedded system such as processing units, interface units and memory. Additionally, different microprocessors like the PowerPC, Microblaze or NIOS have been integrated into the FPGAs as hardmacro or softcore. Different OS are available for theses microprocessors. Diverse applications can run in the OS' user space, for instance an adapted design flow. By using suchlike systems, the complexity of application development decreases due to higher abstraction from the underlying reconfigurable technology.

Following, the different layers of our architecture are briefly detailed. Section 3.1 refers to the NoC layer. In Section 3.2, the OS for dynamically reconfigurable systems is specified. In Section 3.4, the idea of an embedded design flow is investigated and Section 3.3 describes our prototypic simulation environment.

3.1 The Network-on-Chip

The NoC represents the SoCs communication backbone. It is based on asynchronous message passing between the various components, IP cores, and processes in the system. The main building blocks of the NoC are the switches, the Resource Network Interfaces (RNI), the tiles, and the wiring resources required to connect the switches. A tile is a homogeneous and distinct part. It is dynamically reconfigurable. Each tile is connected to one switch. Each switch is connected to four neighbouring switches establishing the communication grid. Every tile connects to the switch fabric via an RNI which is partially fixed (Resource Independent Network Interface, RINI) and partially reconfigurable (Resource Dependent Network Interface, RDNI). This structure is shown in Figure 1 and detailed in [8]. To increase the speed of the NoC and to save costly reconfigurable resources for the tiles, such NoCs are proposed and expected to be a hard-wired component in future FPGAs [6].

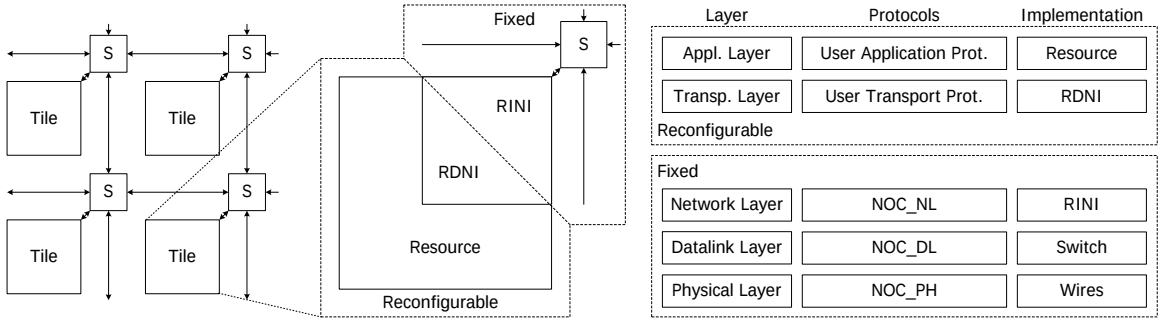

Figure 1. NoC building blocks and protocol stack

Additionally, to represent a high performance communication infrastructure, the NoC supports various user-defined protocols. Therefore, a suitable protocol stack was defined as shown in Figure 1. The structure is similar to the well known and approved ISO/OSI model and TCP/IP stack. The lower layers are implemented in hardware as part of the NoC. They are fixed and provide uniform low level interfaces to each client in the network. Arbitrary user protocols can be

implemented in the reconfigurable layers or in software running on embedded microprocessor cores.

3.2 The Operating System

The OS used for managing dynamic reconfiguration is a modified Debian Linux system, kernel version 2.6.9. The modifications are manifold but rely on standards and approved procedures.

For the OS to access the lower NoC layers, a NoC device driver was already implemented. The same concept as used for Linux network device drivers was adapted to our needs. To allow dynamic reconfiguration, a device driver for the internal configuration access port (ICAP) was implemented, too. The concept of a character device driver was chosen. Both drivers are located in the hardware abstraction layer (HAL).

The kernel space is widened by a set of IP core management instances. Their functional spectrum ranges from IP core registration at the system registry, IP core instantiation and configuration, IP core execution, and scheduling of IP cores to allocation and de-allocation of reconfigurable resources. The latter implicates using mechanisms similar to virtual memory. The difference is that virtual reconfigurable resources are used instead. In [3] and [4] the notion of *virtual hardware* was defined first. Furthermore, the kernel space contains the implementation of a NoC transport protocol. To interface this protocol, the standard BSD (Berkeley Software Distribution) socket API (Application Program Interface) is used. It eases access to the protocol stack, is well-known, and approved.

Application design can be done in an object oriented manner. A computing object covers the individual implementation of an IP core. It does not matter if the functionality is implemented in C++ or in VHDL. The object can only be accessed via its corresponding defined interface. The advantage of an object oriented application design is full abstraction from the underlying hardware. Further, the OS is responsible for which implementation is used in a special case. An IP core can have multiple implementations with diverse attributes. A software implementation, e.g. in C++, represents a decelerated version called decelerator. A hardware implementation of the IP core, e.g. in VHDL or SystemC, represents an accelerator which is mainly used to enhance processing speed. Additionally, the tools for the integrated design flow are located in the user space as one possible application. This design flow is detailed in the next section.

In Figure 2, the levels of our Linux OS are sketched out. In [7], it is presented in detail. Section 3.3 refers to the not yet explained SystemC FPGA model outlined in Figure 2. Section 3.4 details the integrated design flow.

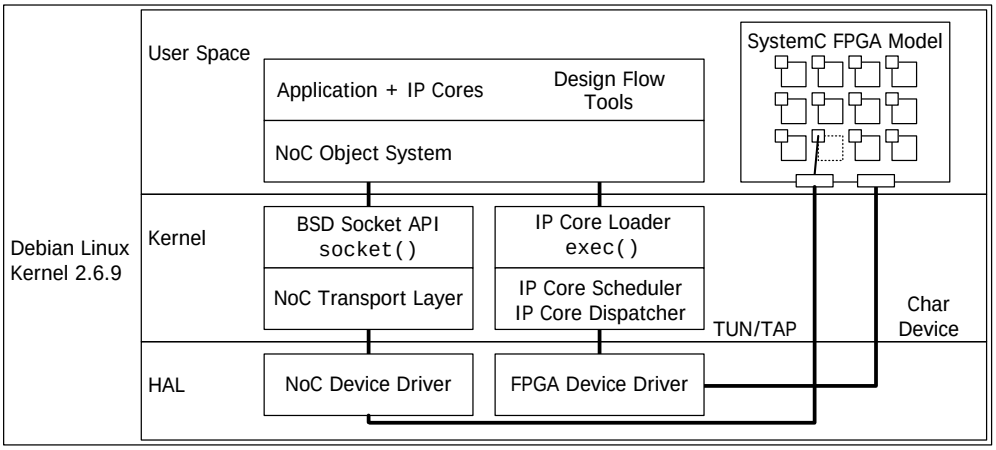

Figure 2. Layered structure of the modified Linux OS

3.3 The Integrated Simulation Environment

In Figure 2, an additional SystemC FPGA model located in the user space is outlined. It is presented in detail in [6]. On the one hand, it serves as simulation and evaluation environment to estimate the systems behaviour and to verify modifications on the system without using a specific hardware. Changes can easily be applied. On the other hand, appropriate tools are largely missing and the use of valuable reconfigurable resources for the NoC fabric is all-consuming. Only a small NoC and a few tiles can be realised on current FPGAs as illustrated in [16].

But even with the availability of adequate technologies and tools, the integration of a functional system model is beneficial in different ways. First, it serves as container for the software implementations of the IP cores, the decelerators. Second, a functional model of the system is used within feedback loops to compare specified and monitored system properties. Thereupon, changes on the systems behaviour can be stimulated. Swapping and relocation of IP cores can be initialised. Or, if it is necessary or beneficial, the systems functionality can be adapted by running the integrated design flow. In [5], an abstract model of the system itself is used for comparison, too. Third, a system simulator is always present. Even after a mission has started, tests can be carried out online with this functional system model. It is cycle inaccurate. But real NoC based systems are cycle inaccurate and non-deterministic, too, because of their asynchronous character. Right now, we can not state whether non-deterministic behaviour is disadvantageous or not in space missions. But future computing systems will definitely work more asynchronous and less deterministic.

3.4 The Integrated Design Flow

Just like any other application, a design flow runs within the operating systems user space as shown in Figure 3. By integrating the complete design flow from the HDL design entry to the device dependent partial bitfiles into the executing system itself, its autonomy is increased. With the help of control mechanisms and a feedback loop, the system verifies specified properties against monitored ones and applies changes on demand without human intervention. Therefore, at least one weighting function is used. In [5], a similar infrastructure for self-adaptive software is presented. This could be of tremendous advantage in outer space missions, when the spacecrafts, rovers or robots are billions of miles away or out of sight. As proposed in [15], the design flow provides compilers for software executables and a synthesis flow for hardware. The executables and bitfiles are stored in a CVS-like library. They can be reused or serve as backup in upset conditions. This is further investigated in Section 4. But a drawback still is the absence of appropriate tools to realise partial reconfiguration.

4 Implications for Space Missions

The introduced architecture of a network-on-chip based SoC reveals several implications for space missions. In the following sections, the problems of capital importance like complexity, autonomy, development time, and robustness are discussed.

4.1 Reduced Complexity

Redundancy, as applied with TMR, is very costly. More silicon devices or at least more reconfigurable resources are needed. Wiring and system complexity explode. Additional resources consume more energy. Extra devices, sockets and printed circuit boards increase the vehicles overall physical payload which is to be kept as small as possible. Therefore, future systems are expected to benefit from virtualisation of hardware. When using the the FPGAs reconfigurable resources as virtual hardware, common techniques as used with virtual memory can be applied and are already implemented in the aforementioned architecture within a Debian

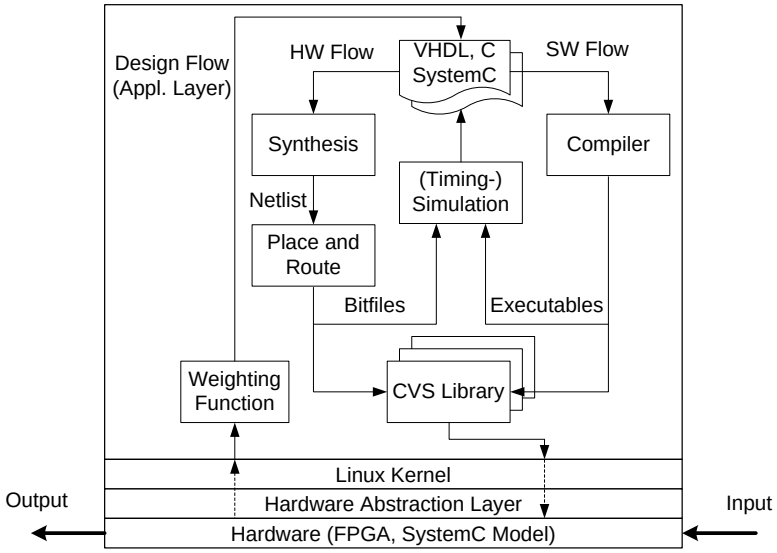

Figure 3. Principle structure of the integrated design flow

Linux. For instance, scientific experiments in space are processed first things first and not in parallel. The necessary functionality can be configured on demand without having every function on-hand at the same time. Another fitting example is everyday portable gadgetry like handhelds, cellular phones, or media players. Theses small devices already provide a broad spectrum of functions that require more and more computational power. When equipped with reconfigurable hardware, the desired application like a media de-/encoder, a crypto core or a 3D accelerator can be instantiated on demand to speed up execution. Due to the small size, a user will never use a cellular phone for two or more different featured applications at the same time. This way, all-in-one devices suitable for every purpose can be designed but are limited to a certain number of in parallel executed tasks.

But complexity is not only reduced regarding the systems physical size. The development process of the software and hardware parts becomes less complex, too, due to the use of object oriented programming languages and the high abstraction from the underlying technology. This is necessary because software and hardware complexity certainly will increase as the functional spectrum broadens. With long-lasting space missions, engineers may even not be able to participate the whole mission from early prearrangements to the final employment because of unforeseeable events. Foreign or new engineers easier become acquainted to the whole system because it is more graspable and concise. In Figure 4, the main steps within the development process are schematised. With the Unified Modelling Language (UML), the overall system structure is defined. Interface description languages (IDLs) and object oriented programming languages like C++ implement the interfaces. The designers will have to do the hardware/software partitioning tasks on their own. With the help of profiling methods the systems bottlenecks can be identified. Then, the functional parts are implemented in appropriate programming languages. Following the compilation, the IP Cores for the reconfigurable hardware and the executables for the embedded microprocessors represent the functionality of the dynamically reconfigurable system. The differences of accelerators and decelerators have already been pointed out in Section 3.2.

In contrast, specialist hardware used in space missions so far offers just minimal tool support and does not necessarily conform to common standards. Only small numbers of devices are manufactured because of the limited market. This complicates the whole development process [11].

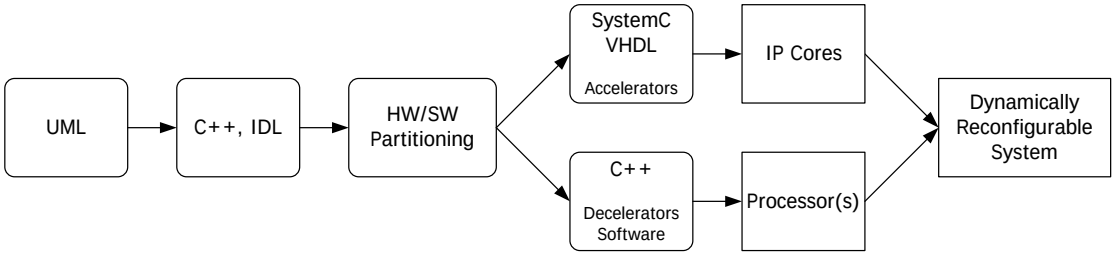

Figure 4. Steps within the development process

4.2 Increased Autonomy

Autonomy is a critical problem within space missions. It is extremely important when line-of-sight obstruction occur between in-field equipment and mission control due to planetary revolution and varying constellations. On the one hand, the higher the degree of autonomy is, the more an engineer has to rely on and depends on the autonomous vehicle out of reach. The autonomous system independently makes decisions in problematic situation to save time and energy resources respectively or at least to prevent from a total loss. Less deterministic behaviour also comes along with increased autonomy. But on the other hand, with dependent and controlled systems, sending commands, receiving acknowledgements, and configuring the executing units in outer space is time consuming and error prone, although the engineer gains more control over the system. So, the bottom line is that communication between mission control and mission equipment and devices is sensitive and prone to errors anyway. Raising the degree of autonomy is beneficial, if the system can still be interrupted and operated from mission control.

Different levels of autonomy exist. First, the vehicles in space are fully managed by mission control. Second, rovers, satellites, and other vehicles may autonomously choose the next pre-scripted instruction sequences for execution in problematic situations without receiving a defined *go* from the terrestrial ground base. Third, with the presented network-on-chip based SoC, parameters or even functionality can autonomously be changed by modifications on the source code level. Therefore, the integrated design flow is used. With help of the on-board system model, the behaviour of the modified system can be estimated before reconfiguring the system. Thus, the vehicles can independently manage themselves within their defined functional scope. Error-prone conversations with mission control are reduced to a beneficial minimum.

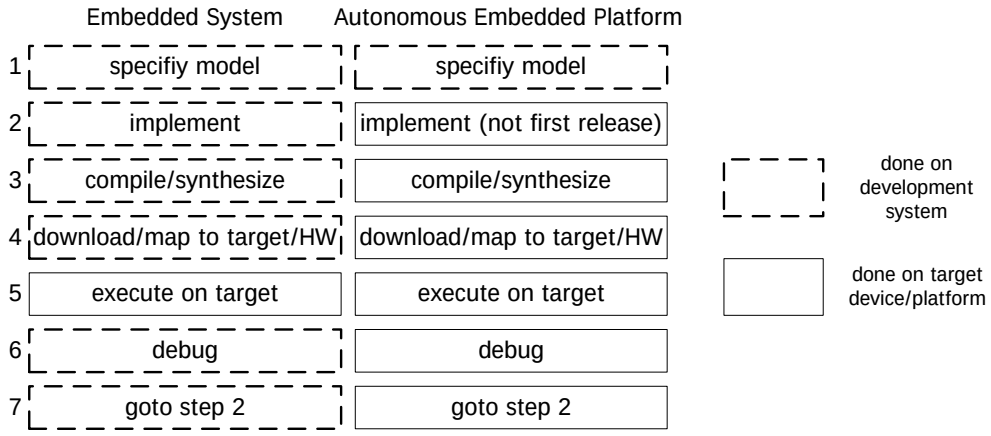

Figure 5. Development Cycle for present (left) and future (right) Embedded Systems

4.3 Shortened Development Cycle

In Figure 5, the state of the art in embedded SoCs and the use of adaptive hardware in autonomous systems as proposed in this paper are compared regarding the development cycle. Theoretically, by combining the reconfigurable device with the corresponding design flow in the same system, no further manipulation from outside is necessary. Only a first release of the initial database must be specified and provided to the target device in *step 1*. The first release is a general model of the embedded systems functional scope. Then, the system evolves by its own stimulus and by the dynamics of the environment to improve, to equalise unbalanced or to fix faulty behaviour. Thus, the system is more independent and flexible. The development cycle and the time to the launch are shortened because specialisation is autonomously achieved through the loop initialised with *step 7*.

The resulting implication for costly and time-critical projects like space missions is the shortened development cycle. On the one hand, applications can be developed from a more abstract point of view due to the object oriented programming paradigm. On the other hand, in-flight simulations using the integrated software system model are possible. If necessary, subsequent updates can be transmitted as plain source code.

4.4 The Systems Life Cycle

To bridge the gap to the fields of organic and autonomic computing, the changes the system independently applies to itself can figuratively be regarded as stages of development within the life cycle of the system. The autonomous platform is intended to improve or at least not to downgrade over time. Most changes may be beneficial. Some changes may be beneficial just for a certain period of time, prejudicial, faulty or occur repeatedly. Other environmental influences can even lead to errors in the hard- and software parts, for instance these in harsh outer space environments. In suchlike cases, it is desirable to switch to a stable system state, to go back to a previous robust configuration, and to ensure at least basic functionalities. Even computation time and energy required for recurrent compile steps within the design flow can be saved. Therefore, previous states must be memorised and minuted. *Concurrent Versions Systems* (CVS) appeal to this scheme and have already been widely accepted in research, development and application. A CVS-like structure can be used to build up a library of bitfiles and executables as proposed in [15].

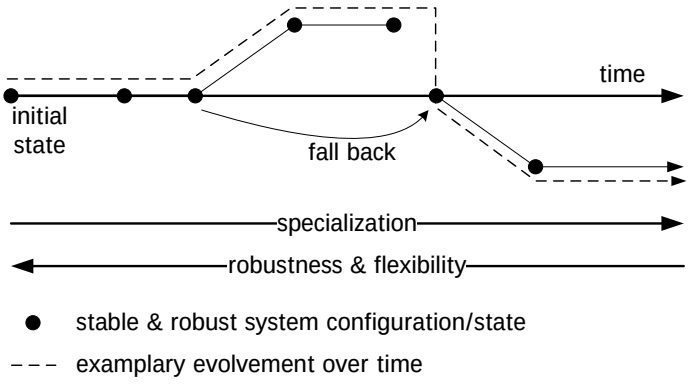

Figure 6. CVS tree showing an evolvable systems life cycle

Figure 6 outlines the life cycle of an autonomous system with a CVS-like tree. Every modification is minuted over time and results in a new system state represented by a point on the main branch (thick line) or on specialised branch levels (thin lines). The initial state represents the first release of the system, its genesis. New branch levels and branchings in the CVS tree indicate

new compiled bitfiles or executables with significant changes affecting the systems functionality. Points on the same branch level may indicate relatively simple parametric changes within a bitfile. The dotted line represents an exemplary systems development over time. But the important aspect is the ability to return to a stable and robust system state in cases of failure. Some redundancy within the system is necessary to secure a smooth transition between two states. Two or more different system states can be compared to continue the better performing branch. Thus, fall back capability in case of failure (but not only then) is provided and performance deteriorating chattering [14] can be reduced. This can be accomplished with the integrated system model previously mentioned.

Another important characteristic of this autonomous system is described in Figure 2, as well. The systems specialisation increases with time as result of the adaptation. In contrast, flexibility decreases *because* of the higher adaptation to the environment. Thus, robustness is decreased. This can be compared with nature. Perfectly adapted individuals will dominate an area as long as the environment is stable. But they are frailer to changes than other less adapted species living in the same area. So, the ability to return to a memorised robust system state is not only beneficial. It is essential for an autonomous embedded system as proposed here. Therefore, a CVS-like library is recommended in this paper to realise self-x attributes as mentioned in the introduction.

5 Conclusion

This paper presented the architecture of a network-on-chip based system-on-chip. First, the systems architecture was briefly described. A network-on-chip serves as communication grid. A modified Linux is responsible for managing partial reconfiguration and abstracts the lower layers to the user. The integrated design flow and the functional system model result in self-x attributes within the system. In the main part of the paper, outer space missions have been used for a case study to point out the implications of our approach on highly sophisticated areas of application. Other potential areas of application are communication and access networks with their ever-changing protocols, parameters, and bandwidth demands or artificial intelligence. The system-on-chip is capable of partial dynamic reconfiguration at runtime. Development time is shortened. Autonomy and robustness increase while complexity decreases or is hidden due to abstraction. But the lack of appropriate tool support and the overhead for the proposed NoC still handicap a concrete realisation on currently available FPGAs. That is why a SystemC model of the FPGA is used instead for validation and verification.

Using reconfigurable devices in outer space missions for the digital computing systems continually improved the quality of the missions outcomes and reduced mission failures until today. But future systems will become more complex, the missions will last longer, and the missions' objectives will be more sophisticated. To meet the increasing demands, reconfigurable logic alone will not suffice. But autonomous embedded SoCs as specified in this paper can find a remedy.

References

[1] Altera Military & Aerospace, www.altera.com/end-markets/military-aerospace.

[2] A. Bartic. Network-on-Chip for Reconfigurable Systems: From High-Level design Down to Implementation. In *Proc. of the Int. Conf. on Field Programmable Logic and Applications (FPL'04)*, Leuven, Belgium, August 2004.

[3] G. Brebner. The Swappable Logic Unit: A Paradigm for Virtual Hardware. In *Proc. of the 5th IEEE Symposium on FPGA-Based Custom Computing Machines*, Napa Valley, California, 1997.

[4] G. Brebner. A virtual hardware operating system for the Xilinx XC6200. In *LNCS*, v.1142, 1996.

[5] D. Garlan, S. H. Cheng, A. C. Huang, B. Schmerl, and P. Steenkiste. RAINBOW: Architecture Based Self-Adaptation with Reusable Infrastructure. *IEEE Computer*, 37(10):46–54, October 2004.

[6] R. Hecht, S. Kubisch, A. Herrholtz, and D. Timmermann. Dynamic Reconfiguration with hardwired Networks-on-Chip on future FPGAs. In *Proc. of the 15th Int. Conf. on Field Programmable Logic and Applications (FPL'05)*, Tampere, Finland, August 24-26 2005.
[7] R. Hecht, S. Kubisch, H. Michelsen, E. Zeeb, and D. Timmermann. A Distributed Object System Approach for Dynamic Reconfiguration. In *Proc. of the 13th Reconfigurable Architectures Workshop (RAW'06)*, Rhodes Island, Greece, April 25-26 2006 (submitted).
[8] R. Hecht, D. Timmermann, S. Kubisch, and E. Zeeb. Network-on-Chip basierte Laufzeitsysteme fuer dynamisch konfigurierbare Hardware. In *ARCS 2004 - Organic and Pervasive Computing, Workshops Proceedings*, Augsburg, Germany, March 26 2004.
[9] E. Horta and J. W. Lockwood. PARBIT: a tool to transform bitfiles to implement partial reconfiguration of field programmable gate arrays (FPGAs). Technical report, Washington University in Saint Louis, Department of Computer Science, July 2001.
[10] M. Huebner, K.Paulsson, and J. Becker. Parallel and Flexible Multiprocessor System-On-Chip for Adaptive Automotive Applications based on Xilinx MicroBlaze Soft-Cores. In *Proc. of the 12th Reconfigurable Architecture Workshop (RAW'05)*, Denver, Colorado, USA, April 2005.
[11] C. W. Johnson. The natural history of bugs: Using formal methods to analyse software related failures in space missions. In *Proc. of the Int. Symp. of Formal Methods Europe (FM'05)*, pages 9–25, Newcastle, UK, July 18-22 2005.
[12] H. Kalte, G. Lee, M. Porrmann, and U. Rueckert. REPLICA: A Bitstream Manipulation Filter for Module Relocation in Partial Reconfigurable Systems. In *Proc. of 19th IEEE Int. Parallel and Distributed Processing Symp. (IPDPS'05)*, 2005.
[13] R. Katz. The Failure of a Small Satellite and the Loss of a Space Science Mission. In *Proc. of the 2002 NASA/DoD Conf. on Evolvable Hardware (EH'02)*, 2002.
[14] M. M. Kokar, K. Baclawski, and Y. A. Eracar. Control Theory-Based Approach of Self-Controlling Software. *IEEE Intelligent Systems*, 14(3):37–45, May/June 1999.
[15] S. Kubisch, R. Hecht, and D. Timmermann. Design Flow on a Chip - An Evolvable HW/SW Platform. In *Proc. of the 2nd IEEE Int. Conf. on Autonomic Computing (ICAC'05)*, Seattle, Washington, USA, June 13-16 2005.
[16] T. Marescaux, A. Bartic, D. Verkest, S. Vernalde, and R. Lauwereins. Interconnection Networks Enable Fine-Grain Dynamic Multi-Tasking on FPGAs. In *Proc. 12th Int. Conf. on Field-Programmable Logic and Applications (FPL'02)*, 2002.
[17] K. Morris. Space Silicon. *FPGA and Programmable Logic Journal*, September 2005.
[18] D. Ratter. FPGAs on Mars. *Xcell Journal*, August 2004.
[19] M. Stumpf. Wind River and NASA - Embedded Development for the Extreme Demands of Space Exploration. *Dedicated Systems Magazine*, pages 25–27, August 2003.
[20] H. Walder and M. Platzner. A runtime environment for reconfigurable hardware operating systems. In *Proc. of the Int. Conf. on Field Programmable Logic and Applications (FPL'04)*, Leuven, Belgium, August 2004.
[21] Xilinx Aerospace and Defense, www.xilinx.com/esp/mil_aero/.
[22] Xilinx Inc., XAPP197: Triple Module Redundancy Design Techniques for Virtex FPGAs.
[23] R. S. Zebulum. Experimental Results in Evolutionary Fault-Recovery for Field Programmable Analog Devices. In *Proc. of the 2003 NASA/DoD Conf. on Evolvable Hardware (EH'03)*, 2002.