# R3TOS: A Reliable Reconfigurable Real-Time Operating System

X. Iturbe[1,2], M. Azkarate[1], I. Martinez[1], A. Perez[1], K. Benkrid[2], A.T. Erdogan[2], T. Arslan[2]

[1]Embedded System-on-Chip Group, IKERLAN-IK4 Research Alliance, Mondragón, 20500, Basque Country
Email: {xiturbe, mazkarateaskasua, imartinez, aperez}@ikerlan.es
[2]System Level Integration Research Group, The University of Edinburgh, Edinburgh EH9 3JL, Scotland, UK
Email: {x.iturbe, k.benkrid, ahmet.erdogan, t.arslan}@ed.ac.uk

**Abstract**: The foundations for building the first Reliable Reconfigurable Real-Time Operating System (R3TOS) are presented. The main objective of R3TOS is to create an infrastructure for coordinately executing specialized hardware tasks upon a reconfigurable FPGA device, achieving the necessary flexibility for both gaining system performance (true hardware multitasking) and tolerating the occurring faults in the underlying chip's silicon at runtime (true fault removal from system). R3TOS is aimed at easing the development of FPGA-based high-performance demanding reliable applications by hiding the complexity of these devices, promoting their use by the whole engineering community.

**Keywords**: Operating System, FPGA, Fault Tolerance, Real-Time, Dynamic Partial Runtime Reconfiguration

## 1. Introduction: The 21st Century FPGAs

21st century Field-Programmable Gate Arrays (FPGAs) are not used for implementing simple "glue logic" functions anymore. They have become a fascinating parallel and distributed extremely advanced compute fabrics with a regular architecture of reconfigurable computational elements and memories, which can be seen as an alternative to the Von Neumann serialized processors in which the computational elements are fixed. In general terms, an FPGA can be considered as a "liquid silicon" which is appropriately "molded" to create the desired functionality. Specifically, SRAM-based FPGAs are able to self-reconfigure the functionality implemented by a part of the computational elements they include while the rest of them are still performing active computation. This capability is commonly called Dynamic Partial runtime Reconfiguration (DPR) and has been significatively improved in Xilinx partial reconfiguration early access tools latest fully supported Virtex-4 family of devices [1,2]. First, the chip fabric is divided into independently reconfigurable units (clock regions), which allow reconfigurable modules to be located in the same column. Second, the new 32-bit wide and 100 MHz running Internal Configuration Access Port (ICAP) permits the reconfiguration to be performed much faster.

### 1.1 Towards Higher Performance

DPR turns the FPGA into a flexible computing device in which it is possible to online configure different custom architectures, each of them specialized for every type of computation to be done. While a program defining instructions sequentially customize the data-path of the Von Neumann processor in order to obtain the best performance as possible from a general purpose computation structure, a partial bitstream configures the functionality assigned to a specific region of the FPGA by defining the required architecture at logical level. Specialized hardware architectures are thus "molded" on the FPGA silicon at runtime, leading to a new computation paradigm which crosses the hardware/software boundaries, combining the flexibility of software with the speed of hardware.

Based on software-like flexibility, the use of FPGA based computers able to schedule their own workload is a natural tendency towards high-performance, as occurred in the software field some decades ago. This time, however, computation tasks are to be executed in a much more appropriate way than when being executed upon a single serialized processor whose architecture is rigid at logical level. On the other hand, hardware-like computing in space leads to massive parallelism exploitation and extremely efficient execution and thus, hardware multitasking arises as a certain reality. In this way, it is possible to circumvent Pollack's rule, which states that the increase in performance of a sequential processor is only about the square root of the increase in its complexity [3]. Hardware tasks can be allocated on the FPGA, executed and finally replaced by other hardware tasks, leading to a continuous stream of input operands, computation and output results, which combines the computation both in space and in time.

Hence, as firstly proposed by Brebner, there is a role for a Reconfigurable Operating System (ROS) in coordinating the concurrent hardware tasks execution as well as making FPGA computational underlying resources easy to be used and shared, by managing them on behalf of the user application [4]. That is, the underlying hardware is "virtualized" and thus, every additional computational elements included in the FPGA are potentially exploitable by

the user application by means of the ROS, leading to increase the computational power.

### 1.2 Towards Higher Reliability

Besides performance, reliability is another aspect also benefited from DPR, which opens the doors to new more advanced possibilities for implementing fault tolerant distributed operating systems some decades after they were firstly conceived (e.g. TUWien developed Mars OS in 1988 [5]). The main idea of these OSs is the replication of tasks execution in different, independent and redundant components, which interact only through an intercommunication network. In this way, as long as any of these redundant components can operate, the required service can be maintained. Additionally, Mars OS was intended to be maintainable since any redundant component could be removed (for repairing purposes) and later reintegrated in the system.

FPGAs allow the migration from the aforementioned late 80's federated distributed systems to System-on-Chip (SoC) integrated architectures, in which replicated hardware tasks are separately yet simultaneously executed on the same chip, being only connected through an appropriate Network-on-Chip (NoC) which ensures no interference will occur between tasks [6]. Opposite to the rigidness of the limited number of redundant components integrable in a federated system, which involved an external intervention in order to recover from faults (usually consisting in component replacement), the massively replicated versatile computational resources available in an FPGA can be autonomously reconfigured for circumventing the faults without requiring any external maintenance [7]. Consequently, by using self-reconfigurable FPGAs a true fault tolerant (not only maintainable) ROS can be developed, able to dynamically allocate every scheduled hardware tasks for execution to non-damaged computational resources. Hence, the system is able to autonomously adapt its own architecture "on-the-fly" in order to overcome fault effect and ultimately, maintain the required service. Likewise, soft-errors caused by radiation can be automatically corrected by performing a *scrubbing* in the configuration memory [8].

These features are especially attractive when systems are difficult to access and operate on harsh environments which induce faults on them. Nowadays these kind of scenarios may be found in deep space exploration, remote sensing and military applications, but in time, the autonomy bringing benefits may also prompt adoption by the commercial sector.

Real-time is another important challenge in high reliability demanding applications. The correctness of a system usually depends not only on the logical correct computation of results, but also on temporal correctness. In this way, precised time constraints must be met, usually dictated by events from its environment. Since FPGA-based computers can fastly process many computations in parallel, they show very short latencies when responding to environmental events and thus, they can bring closer real-time behaviour in the order of milliseconds.

Whether by meeting real-time constraints (temporal correctness) or by keeping the system fault-free at every time (logical computation correctness), DPR permits to step up to the Safety Integrity Levels (SIL) demanded by current international standards (e.g. IEC-61508 [9]).

### 1.3 R3TOS: Universalizing the Reliable High-Performance

However, despite the very promising and attractive possibilities dynamically and partially reconfigurable FPGAs offer in both performance and reliability aspects, currently they are yet to be widely adopted. One reason is the lack of a consolidated and intuitive support for the development, management and execution of hardware tasks based reliable applications. In fact, FPGAs flexibility, which is the key for the innovative possibilities they deliver, is also the cause of the appearing difficulties when designing with them. This situation puts FPGAs at a disadvantage compared to existing abundance of software approaches. Consequently, the easiness for developing this kind of applications on FPGAs must be promoted in order to enable the exploitation of the powerful capabilities these devices deliver by the whole engineering community, especially by those who are not familiar with hardware related issues. After all, software high-level approaches can only mask occurring faults in the underlying silicon substrate, while dealing with them at logical level permit to definitely remove the fault sources from the system [10].

## 2. The R3TOS Approach

The objective of the **Reliable Reconfigurable Real-Time Operating System (R3TOS)**, the first proposed ROS that exploits FPGAs reconfigurability for fault tolerance purposes, is *to provide the necessary support without incurring in additional design costs for continuously adapting system architecture at runtime in order to gain performance while ensuring real-time behavior and tolerate occurring faults in the system*. Thanks to it the application developer is only in charge of defining and implementing the required functionalities as hardware tasks, which are then efficiently managed and executed by R3TOS in a reliable way. This level of abstraction provides the application developers with a "software look and feel" easy-to-use FPGA

based advanced computer, enabling them to take advantage of the ever increasing complexity of state-of-the art reconfigurable devices in order to deal with the exigent requirements modern applications demand and thus, contributing to reduce currently existing productivity gap.

### 2.1 Foundations for R3TOS

The proposed architecture for R3TOS is depicted in Fig. 1. It consists of eight different layers, which are to be executed upon both custom hardware and a general purpose processor (e.g. PowerPC, MicroBlaze or preferably LEON3-FT):

- *COTS Real-Time Picokernel,* which offers a well-known interface for software developers, coordinates software threads execution and gives support for intertask communication and synchronization. It runs upon the general purpose processor.
- *Real-Time Scheduler*, which is aimed at coordinating the access to the ICAP port in an efficient way and assign the appropriate execution starting times to the hardware tasks.
- *2-D Allocator*, which is responsible of allocating the hardware tasks to the non-damaged computational resources available on the FPGA by generating the corresponding configuration partial bitstreams for them.
- *NoC Manager*, which updates the routing tables in the NoC switches according to hardware tasks locations at every time.
- *Dynamic Router*, which is responsible of dynamically creating communication links among the allocated hardware tasks and the NoC interfaces (NIs).
- *Diagnostic Unit*, which performs the diagnosis of the system: at functional level, by analyzing the exchanged information through the NoC, and at physical level, by checking the configuration frames read-back from the FPGA's configuration memory.
- *Placer*, which is aimed at controlling the ICAP port while the reconfiguration process takes place.
- *Inter-Device Coordinator*, which is in charge of coordinating the distinct R3TOS instances running in replicated FPGAs, making it possible the hardware tasks migration among different devices.

These layers should be designed including fault tolerance by design features in order to achieve the highest reliability for the R3TOS core itself.

Additionally, a high-level software tool to program the operating system should be provided. This tool should include a friendly environment in which program the hardware tasks and should cover the later synthesis and download of the generated partial bitstreams for the tasks as well as the initialization of the corresponding variables in R3TOS.

### 2.2 The Architecture of R3TOS

Xilinx XC4VLX200 FPGA is the most appropriate device for implementing an R3TOS-based system as it is divided into 12x2 independently reconfigurable clock regions and includes two clearly differentiated regions: While there are 42 adjacent CLBs in the central region of the device, the exterior region includes both 3-4 BRAMs and 16 CLBs (See Fig. 2). Since BRAMs are necessary for implementing buffers in the NoC interfaces as well as the routing tables in the NoC switches, the exterior region of the FPGA is appropriate be used for task inter-communication purposes, being called communication region. On the contrary, the central homogeneous region is used for allocating the hardware tasks to be executed, being called computation region. In this way, every clock region includes a NoC interface, which establishes the border between the communication region and the computation region, several NoC switches, I/O interfaces and the computational resources to which tasks are allocated. It is remarkable that, opposite to currently existing solutions, R3TOS is conceived to be able to merge and later separate again different clock regions according to the shape and size of the hardware tasks to be allocated.

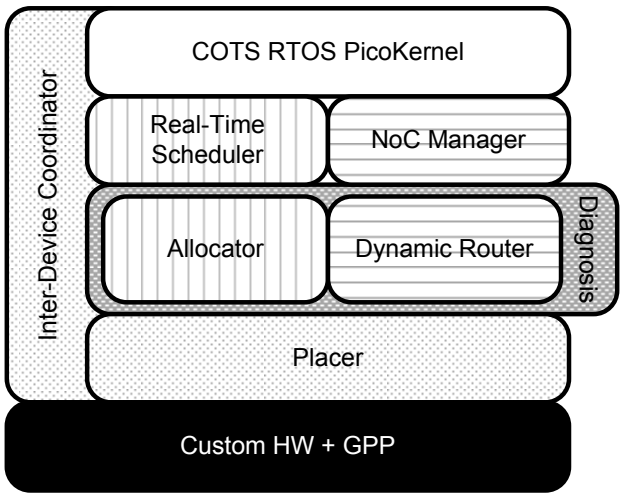


Figure 1: R3TOS layered architecture

The computation region consists of a regular CLBs arrangement, being the regularity the key for increasing tasks allocatability [11]. Consequently, any scheduled task to be executed can be freely

allocated to any specific fault-free position within the computation region and thus, damaged resources avoidance is promoted. When required, the dynamic routing unit online creates a communication link between an allocated task (wherever it is placed in the computation region) and the nearest NoC interface to it. This mechanism permits to circumvent the damaged resources in the proximities of a NoC interface, which otherwise would prevent the utilization of a whole clock region.

On the other hand, the NoC, whose routing tables can be dynamically adapted to the changing locations of tasks, conducts long-distance communication along the chip (e.g. [12]). The NoCs are indeed proven to be the best communication mechanism for coping with faults on high bandwidth reconfigurable systems such as the one here presented [13]. Besides that, since the NoC is the only way of interaction among tasks, it is possible to diagnose the system behaviour by analyzing the value and timing of the information exchanged through it. The diagnosis can be enhanced by evaluating the Error Correcting Codes (ECC) of the configuration frames stored in the FPGA's configuration memory, which can be read-back at runtime.

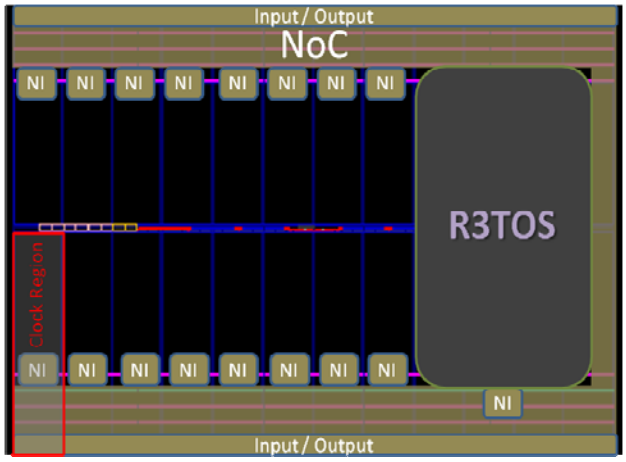


Figure 2: R3TOS-based system implementation in a Xilinx XC4VLX200 FPGA

The tasks are classified according to their communication demand and the criticality of the data they process.

Tasks which demand high amount of data to be interchanged with other tasks or with the environment (e.g. control tasks) are more likely to be placed close to the communication region, in order to ease the information exchange through the NoC. In fact, the only reason for not being allocated next to a NoC interface is the existence of damaged resources in its proximity, which will make tasks to be placed deeper within the computation region, involving the creation of a dynamic route to the nearest NoC interface. On the contrary, tasks which do not require any information to be interchanged (e.g. pure computation tasks) are more likely to be placed opposite to the communication region (See Fig. 3). For this type of tasks, the loading, initialization and result retrieval will be performed through the ICAP port by accessing some registers which are located in specific positions in the task architecture. This schema ensures the best exploitation as possible of the computational resources included in the FPGA device.

Tasks which process critical information are simultaneously executed on different clock regions in order to ensure the logical computation correctness, while tasks which do not perform critical computations are not replicated. Although spatial diversity reduces the probabilities a single fault affects distinct instances of the same tasks, this fact cannot be completely ensured because the FPGA itself is a fault-containment region since a single failure either in the power supply, clock tree or ICAP port could make the entire device useless. As stated in [14], in order to achieve the maximum level of reliability, the system must be partitioned into independent fault-containment regions and thus, tasks must be executed in different FPGA devices which use distinct clock sources and are independently powered (See Fig. 4).

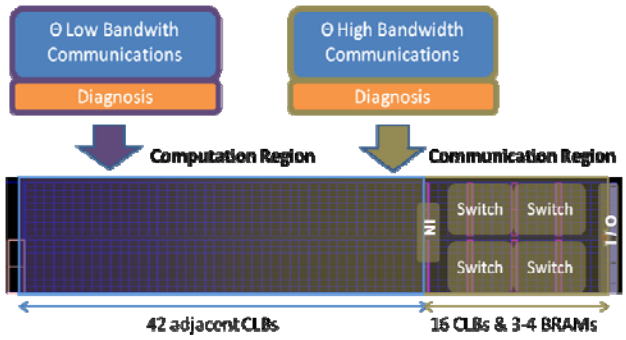


Figure 3: Allocation of tasks depending on their communication requirements

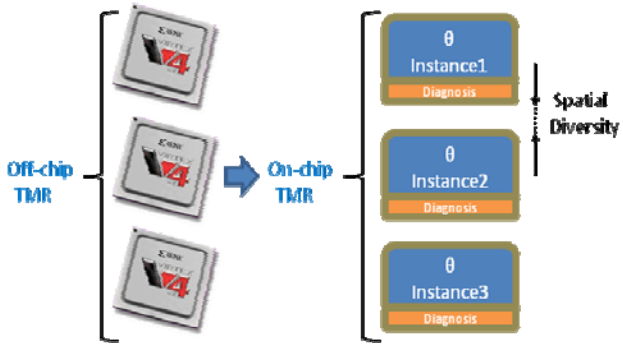


Figure 4: On-chip and off-chip TMR

Fig. 5 summarizes the foundations of the R3TOS-based system explained in this section. At system level, different hardware tasks are allocated and executed. Since $\Theta_1$ hardware task process critical information and demands high bandwidth communications, three instances of it are executed, being connected to the NoC. On the contrary, since hardware tasks $\Theta_2$ and $\Theta_3$ process non-critical information and do not require any data to be interchanged, for each of them a single instance is executed without access to the NoC. The NoC itself constitutes the Communication level, which disseminates the information among tasks in a reliable way. At physical level, it can be observed how the damaged resources are circumvented by means of dynamic routing and fault-aware allocation: The damaged CLB prevents the hardware task $\Theta_i$ to be placed next to the NoC interface. R3TOS is able to manage the entire system through the ICAP port, which enables it to transversally access every system level at runtime.

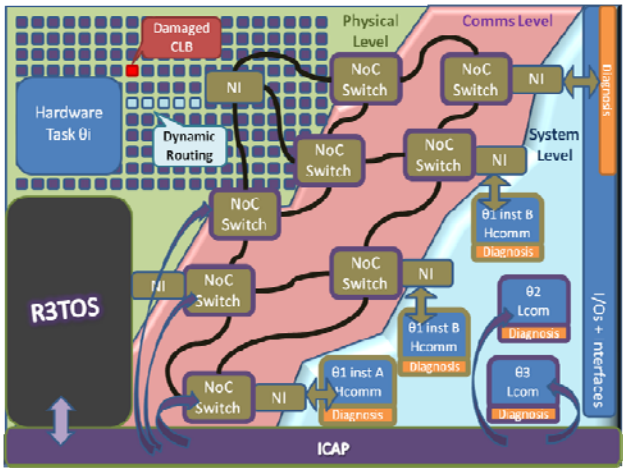


Figure 5: R3TOS based system functional model

## 2.3 Towards Self-Healing in R3TOS

Including fault tolerance by design features in the R3TOS core is not sufficient for enabling its use in ultra-dependable applications. In order to cope with faults directly affecting it, R3TOS core should be able to relocate itself leading to develop a kind of self-healing capability.

Although self-relocation is impossible to be performed by using currently available technology, the ultimate objective can be achieved by including redundant instances of R3TOS in the same chip. Every instance of R3TOS could thus relocate the other instances. Additionally, the R3TOS instances could diagnose each other. For acting so it is necessary to make the ICAP port accessible through the NoC for the R3TOS instances running in the device, as proposed in [15].

As previously stated ICAP is indeed one of the single points of failure of the FPGA and consequently should be meticulously hardened [16].

## 3. Conclusions

The foundations for building R3TOS operating system have been presented in this paper. R3TOS is aimed at increasing FPGA-based systems performance in a reliable way, promoting its exploitation by the whole engineering community towards more advanced and sophisticated applications and ultimately, towards progress.

*Computational Efficiency* is achieved by appropriately managing FPGA's computational elements and coordinating the execution of the computation tasks, in which the application is decomposed (See Fig. 6). The tasks are implemented as specific hardware architectures, which are executed upon the "virtualized" FPGA, being thus possible to execute large applications which require as a whole more computational resources than those incorporated in the FPGA chip.

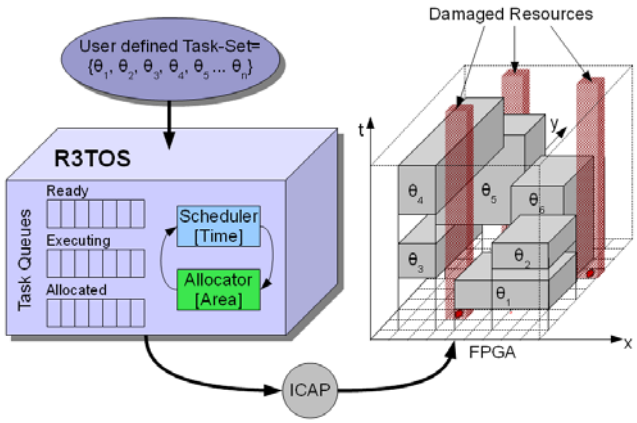


Figure 6: Scheduling, allocating and executing hardware tasks onto a partially damaged FPGA by using the R3TOS approach

*Temporal Correctness* is achieved by using both a real-time scheduler, which ensures every computation to be performed within required deadlines for them, and an appropriate NoC, which ensures real-time communication among the tasks in execution.

*Logical Computation Correctness* is achieved by avoiding the use of damaged resources at logical level and by executing spatial diverse replicated hardware tasks, which ensure the availability of the system. Furthermore, the hardware redundancy saves at least one "logic level" over conventional higher-level software redundancy, achieving reliability without sacrificing the performance.

Hence, by adding R3TOS abstraction layer:

- FPGA related technological complex aspects are hidden, giving to the FPGA a ”software like look and feel” and insulating applications from architectures.
- The required autonomy for dealing with occurring faults in the silicon substrate is achieved once the system has been launched and it is operating out of reach of the designer.
- Device’s lifetime is prolonged and the impact of aging degradation on performance is minimized.
- The direct, efficient and reliable translation of electronic advances into system performance improvement is promoted as well as the reusability of tested circuitry, speeding up development cycles and shortening time-to-market.

Herein presented ideas are next to be implemented, making R3TOS a reality. The authors are opened to any type of collaboration with institutions and organizations sharing similar objectives.

## 4. References


[1] Xilinx Inc.: “*Early access partial reconfiguration user guide*”, UG208, 2008.

[2] P. Lysaght, B. Blodget, J. Mason, J. Young, and B. Bridgford: “*Invited paper: Enhanced architectures, design methodologies and CAD tools for dynamic reconfiguration of Xilinx FPGAs*”, In Proc. of the International Conference on Field Programmable Logic and Applications, pages 1–6, 2006.

[3] S. Borkar: “*Thousand core chips: a technology perspective*”, In Proc. of the annual Design Automation Conference, pages 746–749, 2007.

[4] G. Brebner: “*A virtual hardware operating system for the Xilinx XC6200*”, Proc. of the International Conference on Field Programmable Logic and Applications, pages 327– 336, 1996.

[5] H. Kopetz, A. Damm, C. Koza, M. Mulazzani, W. Schwabl, C. Senft, and R. Zainlinger: “*Distributed fault-tolerant real-time systems: the Mars approach*”, IEEE Microelectronics Journal, volume 9, number 1, pages 25–40, 1989.

[6] R. Obermaisser, C. E. Salloum, B. Huber, and H. Kopetz: *The time-triggered System-on-a-Chip architecture*”, In Proc. of the IEEE International Symposium on Industrial Electronics, pages 1941–1947, 2008.

[7] D. P. Montminy, R. O. Baldwin, P. D. Williams, and B. E. Mullins: “*Using relocatable bitstreams for fault-tolerance”,* Proc. of the NASA/ESA Conference on Adaptive Hardware and Systems, pages 701–708, 2007.

[8] M. Berg, C. Poivey, D. Petrick, D. Espinosa, A. Lesea, K. Label, M. Friendlich, H. Kim, and A. Phan: “*Effectiveness of internal versus external SEU scrubbing mitigation strategies in a Xilinx FPGA: Design, test, and analysis*”, IEEE Transactions on Nuclear Science, volume 55, number 4, pages 2259–2266, 2008.

[9] IEC-61508: “*Functional safety of electrical / electronic / programmable electronic safety-related systems”,* 1st Edition, 1998.

[10] A. Avizienis: “*An immune system paradigm for the design of fault tolerant systems*”, In Proc. of the European Dependable Computing Conference on Dependable Computing, pages 81–83, 2002.

[11] P. Sedcole, B. Blodget, T. Becker, J. Anderson, and P. Lysaght: “*Modular dynamic reconfiguration in Virtex FPGAs*”, IEE Proceedings on Computers and Digital Techniques, volume153, number 3, pages 157–164, 2006.

[12] B. Ahmad, A. T. Erdogan, and S. Khawam: “*Architecture of a dynamically reconfigurable NoC for adaptive reconfigurable MPSoC*”, In Proc. of the NASA/ESA Conference on Adaptive Hardware and Systems, pages 405–411, 2006.

[13] B. Osterloh, H. Michalik, and B. Fiethe: “*SoCWire: A robust and fault-tolerant Network-on-Chip approach for a dynamic reconfigurable System-on-Chip in FPGAs*”, In Proc. of the International Conference on Architecture of Computing Systems, pages 50–59, 2009.

[14] A. Hopkins, T. Smith, and J. Lala; “*FTMP: A highly reliable fault-tolerant multiprocessor for aircraft control*”, In Proc. of the IEEE, volume 66, pages 1221–1239, 1978.

[15] C. Schuck, B. Haetzer and J. Becker: “*An interface for a decentralized 2D-reconfiguration on Xilinx Virtex FPGAs for organic computing*”, International Journal of Reconfigurable Computing, 2009.

[16] J. Heiner, N. Collins and M. Withlin: “*Fault Tolerant ICAP Controller for High-Reliable Internal Scrubbing*”, In Proc. of the IEEE Aerospace Conference, pages 1–10, 2008