A ROADMAP FOR AUTONOMOUS FAULT-TOLERANT SYSTEMS

X. Iturbe\textsuperscript{1,2}, K. Benkrid\textsuperscript{1}, T. Arslan\textsuperscript{1}, I. Martinez\textsuperscript{2}, M. Azkarate\textsuperscript{2}, and M. D. Santambrogio\textsuperscript{3,4}

\textsuperscript{1} System Level Integration Research Group, The University of Edinburgh
\{x.iturbe, k.benkrid, t.arslan\}@ed.ac.uk
\textsuperscript{2} Embedded System-on-Chip Group, IKERLAN-IK4 Research Alliance
\{xiturbe, imartinez, mazkarateaskasua\}@ikerlan.es
\textsuperscript{3} Dipartimento di Elettronica e Informazione, Politecnico di Milano
marco.santambrogio@polimi.it
\textsuperscript{4} Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology
santambr@mit.edu

ABSTRACT

An Autonomous Fault-Tolerant System (AFTS) refers to a system that is able to configure its own resources in the presence of permanent defects and spontaneous random faults occurring in its silicon substrate in order to maintain its functionality. This work analyzes how AFTS could be built, specifically focusing on hardware platform dependant issues, and gives an overview of the state-of-the-art in this field, which is still in its infancy. Three technological levels are used for classifying the research efforts conducted to date. By describing the current state-of-the-art and the constraints imposed by current technology, this work tries to envision future trends towards the ultimate objective of achieving a fully-adaptive system capable of modifying its architecture on-the-fly as needed. Finally, the general structure and organization of a Reliable Reconfigurable Real-Time Operating System (R3TOS) is presented. This OS aims at making the aforementioned adaptability easily exploitable by future commercial applications.

Index Terms— Fault-Tolerance, Partially Reconfigurable Systems, Runtime Reconfiguration, Online Routing, Module Relocation, Hardware Virtualization

1. INTRODUCTION

In today’s world, electronic systems are everywhere, as they become an integral and ubiquitous element of our lives. This trend is likely to continue in the future, as the amount of embedded functionality continues to increase and consequently, electronic systems become more powerful and less expensive. However, the drawback of advanced process technology is a greater susceptibility to static or transient upsets motivated by radiation (even at ground level) or electromagnetic interference, and also to permanent faults due to component aging, electromigration and dielectric breakdown [1]. These faults, if left untreated, might cause system failure, involving catastrophic consequences in safety critical applications and enormous economical losses in many other types of applications. Hence, the need for fault-tolerant systems, firstly introduced by Avizienis in 1967 [2], is currently stated amongst the long term grand challenges of the International Technology Roadmap for Semiconductors (ITRS).

Commercial SRAM-based Field-Programmable Gate Arrays (FPGAs) make it possible to deal with faults and defects in a more flexible way than mask-programmable devices do. Furthermore, Xilinx FPGAs support Dynamic Partial runtime Reconfiguration (DPR), which permits to download new configuration data at runtime so as to dynamically reconfigure a particular portion of the device while the rest of it is still operational [3]. This capability allows for changing system functionality and architecture on-the-fly, overall or partially, opening the doors for new more advanced fault-tolerance techniques inspired by the best fault-tolerant systems ever known: living beings. While the failure of a single transistor in an electronic system often has critical consequences, biological systems are constantly subject to similar or often much more severe failures, and usually they continue to operate unaffected.

Although the flexibility, adaptability and autonomy seen in biological organisms have little comparison with electronic systems regularity, strictness and human-dependance, current FPGAs with DPR feature are thought to be the most suitable low-cost devices in order to shorten the gap between the capabilities developed by living beings and those achievable by electronic systems. Since the cellular organization’s inherent flexibility, and the developed autonomous mechanisms which exploit it, are the key for biological self-healing in living beings, it is reasonable to argue that the more flexible and au-
tonomous reconfigurable electronic systems could be, the better the fault-tolerance levels they could achieve.

This paper analyzes the fault-tolerance viewpoint of autonomous computing. It presents a survey of the most remarkable advances within this promising field and it aims to find the frontier between what has already been done and what is to be done, establishing the starting point for the way leading to building an Autonomous Fault-Tolerant System (AFTS). Being the most flexible low-cost devices, we have identified dynamically and partially self-reconfigurable FPGAs to be the most appropriate devices for achieving autonomy in future commercial applications. In order to ease the exploitation of the flexibility of current FPGAs for reliability and performance purposes, we present the Reliable Reconfigurable Real-Time Operating System (R3TOS) approach as well.

After the context definition proposed in Section 2, we define in Section 3 three different technological levels for FPGA-based AFTS. These levels are used in Section 4 to propose a classification of the most relevant research efforts reported in the technical literature. Furthermore, it also describes the scenario defined by the current state of reconfigurable technology and points out to the main challenges in this field towards higher levels of autonomy. In Section 5, we present a brief overview of the R3TOS approach, responsible for managing system resources appropriately in order to cope with occurring faults in the silicon substrate after system deployment. Concluding remarks are finally presented in Section 6.

2. AUTONOMOUS FAULT-TOLERANT SYSTEMS

2.1. Context Definition

It is well known that in living beings some complex and distributed processes (that autonomously and unconsciously run in every cell that compose multicellular organisms) are responsible for the coordinated management of this flexible structure in order to maintain the organism’s overall functionality intact when it is degraded by disease.

Important efforts have been done during the last years within the Immutronics [4] and Embryonics [5] European projects in order to create an “electronic tissue” to be used for building distributed systems with biological-like self-diagnosis, self-replicating and self-repairing capabilities. All the people involved in these projects agree in stating that the ideal architecture for building AFTSs would include several independently and dynamically reconfigurable areas. In fact, the desired architecture has been implemented as ASICs (e.g. POEth [6], CONFETTI [7]), which incorporate a 2-D array of “electronic cells” that are architecturally equivalent but can be individually reconfigured in order to implement a different functionally at runtime. This makes it possible to carry out a “cell substitution” process, which is indeed one of the key mechanisms of biological fault-tolerance. However, the high-cost and the lack of tools and support have stopped the adoption of these ASICs in the commercial market.

Being the most flexible commercial devices which are affordable for a wide range of applications, FPGAs are set to play a major role in the development of fully exploitable AFTSs in the commercial industry. The most common and general FPGA structure consists of an array of Configurable Logic Blocks (CLBs), Input-Output Blocks (IOBs), clock resources and routing channels. Modern devices also include some special resources (e.g. Block-RAM memories and DSPs), leading to more powerful and heterogeneous reconfigurable architectures. The functionality assigned to the logic resources included in the FPGA and their interconnection is defined by a configuration bitstream, which is stored into device’s internal configuration memory. For modern Xilinx FPGAs the bitstream is divided into the so-called configuration frames, which are the smallest amount of configuration information that can be accessed in the configuration memory and define thus the configuration granularity of the device. Since a configuration frame configures the resources included within a column that spans the whole height of a fabric clock region, this is the minimum chip’s logic area modifiable when using DPR. Modern Xilinx FPGAs ease the access to the configuration memory by including an Internal Configuration Access Port (ICAP) which removes the need for external components that control the reconfiguration process. However, FPGA’s low-cost comes with a price. First, the parallelism in the reconfiguration is sacrificed. There is an unique port to access the configuration plane of the FPGA and thus, different parts of the system must be sequentially reconfigured. Second, the reconfiguration speed is limited. In order to perform a standard reconfiguration, milliseconds are typically required. Finally, on the contrary of the aforementioned cell-based ASIC circuits, the fine-grained architecture of Xilinx FPGAs is not specifically oriented to build any biologically inspired fault-tolerance mechanisms.

In general terms, an FPGA can be considered as a ‘liquid silicon’ (subject to manufacturing defects existence and spontaneous faults occurrence) in which the logic resources are appropriately configured to create the desired system, consisting on several processing elements and the corresponding communication channels among them. This configuration, as well as dynamic adjustments to it, should occur automatically (without any external human intervention) to better adapt to changing necessities, e.g. faults, in order to progress towards the observable autonomy in living beings. But... Who has to mold the ‘liquid silicon’ of the FPGA to create a functional design at every time and for every faulty configuration?

When answering this question one finds out the desirable autonomy capability. At design phase, the designer is the responsible for creating a functional system using device’s resources, but these resources will be unpredictably affected by faults. Consequently, much later after the designer has finished creating the system, the ‘liquid faulty silicon’ should be
be re-molded, even in applications in which the creator has no access to the creation; think about NASA’s planned long duration deep space exploration missions in which communication with Earth is too slow and too limited for it to be used to remotely cope with unpredicted dangerous situations in real-time. Moreover, the idea of (accessible) systems able to maintain their functionality by themselves, is very attractive due to the subsequent reduction in maintenance costs [8].

Hence, the research question is: how to autonomously manage chip’s logic resources at runtime in order to maintain system functionality in the presence of faults.

2.2. Structure and Basic Definitions

An AFTS is able to answer the above research question, being distinguished from a conventional fault-tolerant system since it is able to deal with the occurring faults at runtime which are unpredictable at design time. Thus, an AFTS can avoid fault induced degradation effects, by appropriately managing its logic resources.

The ideas presented in this paper are mainly inspired by the work previously done by Steiner and Athanas, who defined a roadmap for building an autonomous computing system and developed a proof-of-concept implementation [9]. Steiner and Athanas aimed to design an autonomous system in the broadest sense of the word, that is, a system able to grow, reason, learn and adapt. Consequently, the proposed roadmap extends along physiological or technological issues, behavioral aspects and cognitive matters.

Regarding fault-tolerance, we propose three technology-dependant foundation levels upon which to build an FPGA-based AFTS, able to adapt its overall architecture in response to faults in such a way that the system is kept fault-free at all times (See Fig. 1). These levels could be autonomously managed by R3TOS, which would hide FPGA related technological complex aspects and offer a reliable platform upon which applications could run. Our hardware focussed interpretation matches Avizienis’ proposal, who suggested the “Electronic Immune System” concept in [10]. According to his conception, the immune system of the body (analogous to hardware) is a major protective mechanism and it is completely independent from the cognitive processes (analogous to software).

As depicted in Fig. 1, the capability of self-diagnosis is mandatory when building an AFTS. In fact, the three aforementioned levels require the ability for locating the damaged resources in order to circumvent them when reconfiguring the new fault-free architecture. Current Xilinx FPGAs address this issue and incorporate a built-in diagnosis block, called Frame ECC, which automatically checks the ECC bits included in the configuration frames as the bitstream is read-back from the configuration memory. In the case that an upset has occurred, this check permits to know the position of the corrupted bit within the bitstream, being immediate to locate the affected resource.

3. TECHNOLOGICAL LEVELS FOR XILINX FPGAS

The aforementioned Technological Levels (TLxs) determine the achievable effectiveness when coping with faults. Fig. 2 is a symbolic representation of these levels. It is aimed at showing the way each of them generates fault-aware configurations for the AFTS. Higher technological levels involve a more effective use of device’s resources at finer granularity, making it possible to gain flexibility and thus, enhancing the capability for generating new configurations which circumvent faults. Consequently, the higher the technological level, the higher the fault-tolerance capability.

3.1. TL0: Minimals

The reconfiguration resources as well as diagnosis mechanisms of TL0 systems are remotely managed by the designer, who directly access both the reconfiguration port of the reconfigurable device (ICAP or SelectMAP for Xilinx FPGAs) and the diagnosis circuitry (Frame_ECC Primitive for Xilinx FPGAs). The designer is thus responsible for implementing both the architecture exploration and the system’s architecture update. Hence, TL0 systems are not aware of their own architecture and consequently they are not autonomous, but
incorporate the minimals for building AFTS upon them. That is, TL0 require system remote maintenance after having been launched. An example of TL0 is NASA developed Reconfigurable Scalable Computing (RSC) project [11]. The purpose of this project is to reconfigure the devices used in space missions remotely from Earth at runtime, adapting them to changing conditions or reassigning their logical resources as required. Likewise, in [12] the system is remotely re-synthesized in order to circumvent the damaged resources.

3.2. TL1: Self-Reconfiguration

TL1 systems play a minor role in their autonomous adaptation since they are able to update their configuration without any external intervention. Although these systems are not able to generate new configurations on their own, the continuous self-reconfiguration with the configurations provided by the designer allows for fixing radiation induced bit-flips which do not involve permanent on-chip damage (e.g. scrubbing technique [13]). However, permanent damage in the chip can only be avoided if any of the configurations provided by the designers permit to do so.

The effectiveness can be improved by dividing the system into several slots and defining several partial configurations for each of them. This partition makes it possible to split the system’s complete configuration into several partial pieces, increasing the number of possible combinations among them and consequently increasing the number of faulty situations that can be overcome [14]. However, the fault-tolerance capability is limited to those faulty situations considered at design phase due to the loss of versatility when using the stored partial configurations: the slots are constrained to a specific location and the boundaries cannot be changed at runtime. Furthermore, multiple and nearly identical partial configurations must be stored. Thus, the more faulty situations are taken into account, the more configurations need to be stored and therefore, the higher storage required.

An example of TL1 can be found in [15]. In this work, up to three different partial configurations are used that can reconfigure each slot of the system. In addition to this, the system executes fault correction scrubbing as well. Likewise, in [16] a survey on slot-based systems for fault-tolerance is presented, while in [17] an approach which takes advantage of the inherent architecture diversity of distinct functionality modules is reported.

3.3. TL2: Dynamic Module Relocation

The functional modules of TL2 systems are relocatable and thus, they can be freely placed at any 2-Dimensional position within the reconfigurable device. Hence, although the architecture of the modules cannot be modified at runtime, the flexible module relocation makes it possible to generate new system configurations on-the-fly in order to tolerate dynamically occurring faults, even those that were not predicted at design time. Based on a single relocatable module which requires $h_x \times h_y$ resources, the TL2 system is potentially capable of generating up to $(H_x \cdot H_y) / (h_x \cdot h_y) - 1$ extra system configurations for a reconfigurable device which incorporates $H_x \times H_y$ resources. Therefore, TL2 systems do not have to store redundant partial configurations and thus, they do not incur in extra storage requirement. A rigorous (and often complex) system architecture exploration needs to be carried out in order to find the appropriate location for all modules in such a way that damaged resources are not utilized.

Although there is an important background in module relocation, as presented in Section 4.1, its use in the fault-tolerance domain is limited. The work presented in [18] is one of the few research efforts that can be classified in TL2.
3.4. TL3: Online Re-synthesis

TL3 systems are able to autonomously restore their functionality, creating an adapted new configuration at runtime for nearly every faulty situation. By re-synthesizing modules online TL3 systems can generate a potentially infinite number of distinct configurations. The online re-synthesis capability operates at the device’s finest granularity e.g., for TL3 systems the processing elements \( \rho_i \) match with FPGA’s logic resources and the communication channels match with FPGA’s routing wires. Moreover, online re-synthesis permits to skip just the damaged resources and thus, resource waste due to faults is minimal. Conceptually this scenario is similar to the biological cell regeneration in the sense that both the substances that belong to damaged cells and the resources used by faulty modules are reutilized.

As far as we know currently there is not reported any successful implementation of any TL3 system in the technical literature due to existing technological constraints, discussed in Section 4.

4. THE SCENARIO DEFINED BY CURRENT FPGAS

Current FPGA technology imposes some restrictions for achieving the capabilities defined in the aforementioned levels. The online synthesizer and technology mapper required by the TL3 directly faces up to current lack of suitable low-level tools, motivated by the absence of public documentation about proprietary FPGA bitstream formats.

First, the bitstreams are indeed exclusively generated by high level abstraction proprietary synthesis software (e.g. Xilinx PAR and Bitgen tools) [19, 20]. Technically, it would be possible to run such proprietary synthesis software on-chip, but only Kubisch et al. timidly propose this idea in [21]. Furthermore, the time and computational resources required in order to do so make this option unfeasible for those applications in which the observable fault rate is high.

Second, the fine-grained access to the bitstream is nowadays not possible, except to those bits whose function has been previously identified by reverse-engineering [19, 20, 22, 23] or by using Xilinx provided JBits API [24, 25]. While the first two referenced efforts obtain this information by comparing the bitstream changes after specific design modifications, the last three works parse the EDIF-like FPGA fabric netlist file generated by Xilinx Design Language (XDL) tool. The XDL netlist is indeed the only detailed description of the FPGA fabric structure provided by Xilinx in non-proprietary ASCII format. On the other hand, JBits API translates high level instructions into bitstream domain, e.g. CLB-shifting based fault avoiding directives presented in [26]. Unfortunately, JBits API is nowadays deprecated as it does not support any device introduced in the last decade.

An alternative is to use an Evolutionary Algorithm (EA) to blindly explore the fault-affected bitstream unknown space in order to find a configuration for which the FPGA implements the required functionality. The evolutionary blind search, opposite to previously presented guided one, does not depend on any low-level bitstream manipulation tool. It arbitrarily modifies the bitstream of the system and evaluates the behavior of the new circuit. If the change improves the functioning of the system it will be preserved and, in case it is not positive for the system it will be removed.

The autonomous execution of the evolutionary search in the same chip whose bitstream is being analyzed is called intrinsic Evolvable Hardware (EHW). Although EHW fails to take advantage of the diagnosis information, it is reported to be effective with simple circuits whose bitstream is in the order of some hundreds bits long [27, 28]. However, the convergence is highly constrained when using EHW for exploring today’s complex systems involving large solution spaces (up to some Megabits long bitstreams). One of the non-achieved challenges in the use of EHW for fault-tolerance is indeed the fault isolation within the unknown bitstream domain. Note that fault isolation would significantly reduce the bitstream portion on which the evolution process would take place.

Due to these technological restrictions, the authors posit that the highest technological level currently achievable is TL2. Such an autonomous system should be capable of online carrying out the placement and routing of the modules generated by proprietary software tools at design time, which define the granularity of the system itself and set the fault isolation borders. The modules are thus the structural and functional basic units of the autonomous system, which appropriately manages them in order to circumvent the faults. By managing pre-synthesized modules as a whole, there is no need to deal with the fine-grained details of the bitstream.

Fig. 3 classifies the technical literature on FPGA-based AFTS according to the levels introduced in Section 3.

![Fig. 3: Classification of the technical literature.](image-url)
4.1. Online Module Relocation

The relocatability of modules defined at TL2 is based on the Merge Dynamic Reconfiguration method (MDR) [29]. The latter exploits Xilinx FPGAs’ regular structure for allocating a partial bitstream in different arbitrary positions by appropriately managing the configuration frame addresses while the reconfiguration process takes place. That is, the configuration frames are shifted within the full system bitstream. This can be done by using REPLICA-like bitstream filters [30]. Consequently, it is not necessary to store a partial bitstream of the same module for each position where it is possible to be allocated.

Modern reconfigurable architectures bring new innovative possibilities and involve less limitations than their predecessors when partially reconfiguring certain regions of the device at runtime [31]. Opposite to their column-wise configuration frame based predecessors, modern devices are configured by means of clock region-based frames and include separate frames for IOB columns that ease the reconfiguration process. Consequently, the minimum chip’s reconfigurable logic area modifiable when using DPR in these devices is a clock region spanning resource column and thus, any given module allocated in a clock region can be individually modified without interrupting the rest of the modules allocated in different clock regions. These possibilities have been addressed in recent research efforts. In [32] the authors present a method for generating partial bitstreams using dedicated clock resources. This alleviates the Xilinx tools-provoked clock routing inefficiencies when synthesizing relocatable partial bitstreams. Additionally, this method enhances MDR to support local clock domains, allowing the relocatable modules to operate at their own independent clock frequency. However, modern FPGA fabrics show a continuing trend of growing heterogeneity, which is problematic for relocation. In [33] the authors explain the basis for dealing with the heterogeneous resources incorporated in these last generation FPGAs. They could allocate a module in a target position where there were different types of resources from the region for which it was synthesized. Finally, in [34] the authors describe a design method for selecting a synthesis region for the relocatable modules with the objective of optimizing the placement at runtime.

By using herein presented state-of-the-art trends, system flexibility is significantly enhanced when compared with traditionally used Difference based and Module based DPR design flows [35], which lead to only rigid slot-based TL1 systems. The development of this type of systems is fully supported by Xilinx PlanAhead tool [36]. This tool eases the partitioning of the design into slots, checks whether all the design rules are met in the partially reconfigurable system and finally, generates the partial bitstreams for every module allocated in each reconfigurable slot. PlanAhead gives support for establishing the communication interfaces to the slots as well. The communication to and from the slots must be done through the provided bus-macros that serve as the bridge between them and whose location is fixed [37].

In contrast to the existing support for designing TL1 systems, there are still several challenges to be overcome for building fully functional TL2 systems.

1.- Clock signal distribution related concerns: The clock signal must reach all the logic resources to be used when relocating a module (those which are initially configured and the ones which initially are not used). Unfortunately, Xilinx tools do not allow for the management of the logic resources not used in the original system, which indeed defines the scenario on which the module relocation will perform.

2.- Structural non-regularity related concerns: It is desirable to be able to by-pass the existing heterogeneous logic resources in state-of-the-art devices in order to enhance the relocatability of the modules, as shown in Fig. 4. This would require a slight modification of the internal architecture of the modules, something which is not currently supported.

3.- Relocatable partial bitstreams generation related issues: Special cares must be taken with the static routes that inevitably pass through the reconfigurable regions. Since the routes are static, they must operate even while partial modules are relocated. This means that the routing resources used by these routes must be always respected. In the aforementioned MDR technique static routes are identified and preserved by using an exclusive OR (XOR) operator to merge the partial bitstream with the corresponding static configuration [29].

4.- Configuration granularity related issues: The size of the reconfiguration frames constrains the relocatability of the modules. Currently the feasibility of “shifting the configuration bits within a configuration frame” is not proven. This means that although the only condition to be met for horizontal relocation is that the structure of the target region must be compatible with the original one [33], the vertical relocation is constrained by the amount of existing clock regions in the device. Actually the vertical placement for a relocatable module within a clock region cannot be modified at runtime, as depicted in Fig. 5.
4.2. Online Routing

In order to keep the communications of the modules when they are relocated to different positions, the Online Routing capability is required. However, in the contrary to prefixed-routes based communications (e.g. [29], [38]), online routing is not sufficiently supported by current design tools.

Technically, it is possible to directly use FPGA’s routing resources by activating the appropriate Programmable Interconnection Points (PIP) as presented in [23]. In this work, the authors could generate a database with the routing resources information, which is obtained from the netlist file generated by Xilinx XDL tool. By exploiting the repetitiveness of the routing fabric in current Xilinx devices, the routing database’s size is several orders of magnitude smaller than those used by vendor tools. Hence, a “Wire-on-Demand” can be created at runtime up to four times faster than when using Xilinx PAR.

However, since the routing information is vast, complicated and not documented for commercial FPGAs, it is nowadays common to avoid FPGA’s PIP direct management when performing runtime routing and use instead other techniques or routing elements, such as LUT-based multiplexors. In [22] the LUT content which defines all possible selections for these multiplexors are given. By using difference-based DPR it is possible to change these contents at runtime, modifying the communication channels among modules. In order to do so, the exact location of the LUTs on the FPGA must be known, which is done by defining hard-macros. It has also been proposed to use an EA for exploring the communication space defined by multiplexors’ configurations [22].

In order to avoid using the routing resources, in [39] the authors consider a communication route as any other type of functional modules. They store several partial bitstreams associated to different communication channels and retrieve them according to the communication necessities of each module allocation. Following the same principle, in [40] the routing is based on some basic hard-macros (the so-called routing primitives) which implement both vertical/horizontal routing connections and module interfaces by exclusively using CLBs. These routing primitives are adjacentely placed as needed in various slots defined at design time (equivalent to the slots used in TL1) in order to create larger communication channels. This strategy involves thus a scale reduction of the previous approach in which the partial bitstreams associated to the communication channels are defined at the chip’s finest granularity, enhancing their reusability. Although this strategy is appropriate for skipping just the damaged resources, and consequently maximize the amount of non-damaged useful area, the routing possibilities are limited by the slots defined at design time. This limitation can be overcome by using the MDR method [41]. In this work, the routing primitive hard-macros are freely placed as long as their size is a multiple of the allocated modules size.

4.3. Hardening the Reconfiguration Controller

From a fault-tolerance point of view, the approaches presented in the previous sections suffer from the fact that they are based on a central processor that coordinates the entire system by accessing its configuration. Consequently, this reconfiguration controller is a single point of failure since a single fault in it could make the complete system fail.

Aiming at increasing the reliability of the reconfiguration, in [42] the authors present a reconfiguration controller that includes various fault-tolerance by design features, including Triple Modular Redundancy (TMR) and specialized scrubbing for the internal memories. Furthermore, as Virtex-4 devices include two separate ICAP interfaces\(^1\), it is even possible to reconfigure the reconfiguration controller itself, enabling the exploitation of the aforementioned technological levels for this critical part of the AFTS system.

The Digital on Demand Computing Organism (DodOrg) project [43] has a much more ambitious objective: to put the reconfiguration access port at the service of all the components of the system in such a way that several reconfiguration controllers can be included in a system in order to diagnose and recover each other, making the whole system more reliable. Note that this approach is aimed at stepping up to the ideal distributed architecture described in Section 2.1.

5. THE R3TOS APPROACH

The work carried out by the Configurable Computing Laboratory at Virginia Tech. brings the future of AFTS closer. The

---

\(^1\)However, the implementation has the two interfaces share the same underlying logic.
The hardware tasks are automatically allocated in such a way that they can be seen as if they are dynamically adapted to the changing locations of the device although there are sufficient resources available. Ultimately, it could lead to the undesirable situation that a hardware task cannot be executed on the device although there are sufficient resources available.

When targeting the commercial sector, real-time performance is nowadays a mandatory constraint [45]. This is why we have included this feature in the OS we propose. Aiming at increasing the system reliability and performance, on-chip resources are conveniently managed and configured on behalf of the user, as firstly proposed by Brebner in [46]. Modules are swapped in and out from the device according to the computation necessities and device’s state at every time and thus, they can be seen as hardware tasks. In the R3TOS approach the hardware tasks are automatically allocated in such a way that damaged resources in the chip are not used.

The objective of R3TOS is thus to provide the necessary support without incurring in additional design costs for autonomously adapting system architecture at runtime, in order to tolerate occurring faults as well as to gain system performance while ensuring real-time behavior [47]. Whether by meeting the real-time constraints of the task executions (temporal correctness) or by keeping the system fault-free at all times (logical computation correctness), R3TOS permits to step up to the Safety Integrity Levels (SIL) demanded by current international standards (e.g. IEC-61508). Thanks to it, the application developer is only in charge of defining and implementing the required functionality as a set of hardware tasks (creation of the system), which are then efficiently managed and executed by R3TOS in a reliable way (management of the system).

5.1. Functional Structure of R3TOS

The main parts of R3TOS are the Scheduler and the Allocator, as shown in Fig. 6. The real-time scheduler coordinates access to the ICAP in an efficient way and thus, it assigns the appropriate execution times to the hardware tasks [48]. The allocator is aimed at deciding the best placement for every hardware task selected to be executed. That is, the scheduler dictates the chronological order of execution of the triggered tasks (time domain analysis) and the fault-aware allocator maps them onto the available logic resources at the time (area domain analysis).

Hence, the allocator must deal with the technological levels presented in Section 3. Most of the research efforts carried out to date resort to splitting the reconfigurable area of the FPGA into TL1 reconfigurable slots in which the hardware tasks are allocated (e.g. [49, 50]). However, this lack of flexibility leads to too much waste of resources each time a fault occurs (note that a single fault could disable a complete slot) and reduces the amount of exploitable computational power of the device (note that unused resources within a slot are not usable by other tasks). Ultimately, it could lead to the undesirable situation that a hardware task cannot be executed on the device although there are sufficient resources available.

The promising advances reported by the Configurable Computing Laboratory at Virginia Tech. encourage us to develop the first TL2-based OS for FPGAs: R3TOS.

5.2. Towards R3TOS Implementation

R3TOS is composed of two parts. The Computational Region refers to the part of the FPGA where the hardware tasks are executed. That is, the region that people from Configurable Computing Laboratory at Virginia Tech. name as “sandbox”. Besides that, a communication infrastructure (e.g. A Network-on-Chip) that conducts long-distance communication along the chip is implemented in the Communication Region. The NoCs are proven to be the best communication mechanism for coping with faults on high bandwidth reconfigurable systems such as the one here presented. Specifically, we are thinking on a highly-flexible NoC whose routing tables can be dynamically adapted to the changing locations of the hardware tasks.

Fig. 6: Scheduling, allocating and executing hardware tasks onto a partially damaged FPGA with the R3TOS approach.
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. 7). 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 for the communication region. On the contrary, the central homogeneous region is used for the 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. Ideally, R3TOS should be able to create a communication link between any allocated task (wherever it is placed in the computation region) and the nearest NoC interface at runtime. The online routing capability permits to circumvent the damaged resources in the proximities of a NoC interface.

![Diagram of R3TOS implementation](image)

**Fig. 7**: R3TOS implementation in a Xilinx XC4VLX200 part

In order to achieve the highest reliability, R3TOS should be distributed along different FPGA chips. Note that an FPGA is a fault-containment region because a single failure either in the power supply, clock tree or the reconfiguration port could make the entire device useless. Consequently, 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. Furthermore, from a high-level perspective it is possible to reconfigure different pieces of the system at the same time and thus, the off-chip distribution of R3TOS could be compared to the on-chip ideal distributed architecture presented in Section 2.1.

### 6. CONCLUSIONS

The bases for building Autonomous Fault-Tolerant Systems (AFTS) have been described in this paper. These adaptive systems are capable of autonomously responding to randomly occurring faults in their substrate, guaranteeing correct functionality despite fault presence. Autonomy should be especially attractive when systems are difficult to access and operate under severe radiation, extreme temperature, high pressure or corrosive 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 benefits brought about by autonomy may also prompt adoption by the wider commercial sector. In fact, fault recovery capability not only permits to build more reliable systems, but also reduces maintenance costs and increases system lifetime while minimizing the impact of ageing degradation.

Being the most flexible low-cost devices, we have identified dynamically and partially self-reconfigurable FPGAs to be the most appropriate alternative for achieving autonomy in future commercial applications. First, we have defined three different technological levels for FPGA-based AFTS, and proposed distinct capabilities for each of them: the higher the technological level, the more autonomous systems we could build. Second, we have summarized the most important approaches adopted up to date with the final objective of building an AFTS and we have later classified them according to the aforementioned levels. The current state of reconfigurable technologies has been presented and the main challenges towards higher autonomy levels has been described.

Finally, we have proposed a brief description of a Reliable Reconfigurable Real-Time Operating System (R3TOS), responsible for managing system resources appropriately in order to cope with occurring faults in the silicon substrate after system deployment. It is aimed at making the capabilities delivered by AFTSs easy to be used by future commercial applications.

### 7. REFERENCES


