Design of Hardened Embedded Systems on Multi-FPGA Platforms

Cristiana Bolchini, Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano, Italy
Chiara Sandionigi, CEA, LIST, Embedded Computing Laboratory, 91191 Gif-sur-Yvette CEDEX, France

The aim of this article is the definition of a reliability-aware methodology for the design of embedded systems on multi-FPGA platforms. The designed system must be able to detect the occurrence of faults globally and autonomously, in order to recover or to mitigate their effects. Two categories of faults are identified, based on their impact on the device elements; i) *recoverable* faults, transient problems that can be fixed without causing a lasting effect, and ii) *non-recoverable* faults, those that cause a permanent problem, making the portion of the fabric unusable. While some aspects can be taken from previous solutions available in literature, several open issues exist. In fact, no complete design methodology handling all the peculiar issues of the considered scenario has been proposed yet, a gap we aim at filling with our work. The final system exposes reliability properties and increases its overall lifetime and availability.

Categories and Subject Descriptors: B.7.3 [**Integrated Circuits**]: Reliability and Testing

General Terms: Design, Reliability

Additional Key Words and Phrases: Fault tolerance, Multi-FPGA, Partitioning

ACM Reference Format:
Bolchini, C., Sandionigi, C. 2013. Design of Hardened Embedded Systems on Multi-FPGA Platforms

1. INTRODUCTION

In the last decades, Field Programmable Gate Arrays (FPGAs) have established themselves as target technology for both fast prototyping and final production of embedded systems. Among the many different families, Static RAM (SRAM) FPGAs are the most frequently used devices, because they offer various advantages with respect to other technologies. In particular, their most attractive feature is the flexibility deriving from the opportunity of re-programming (or reconfiguring) the device. The reconfiguration can be performed on-line, while the device is operating; in this case, we speak of dynamic partial reconfiguration. SRAM-based FPGAs can be reconfigured as many times as desired, by loading the corresponding configuration in the device memory, so that the new/modified functionality can be performed. This, allows the system implemented on the FPGA to be updated/upgraded, also remotely, at a final cost that is quite limited with respect to other technologies, especially when considering low production volumes. For this reason, such devices are currently employed in many domains and their use is being investigated also for critical environments, such as space application ones, where the direct maintenance of the system is a difficult activity.

However, this flexibility is also a source of problems, because the memory storing the device configurations is susceptible to radiation-induced errors [May and Woods 1979; Ziegler and et al. 1996] that may corrupt the content, thus modifying the implemented functionality. As a consequence, SRAM-based FPGAs cannot be adopted straightforwardly as a platform, but fault mitigation techniques need be introduced to obtain a dependable system. Nevertheless, the technology is more convenient with respect to alternative solutions characterized by higher costs, such as Application Specific Integrated Circuits (ASICs) and radiation-hardened FPGAs. In particular, among the radiation-hardened devices, there are some families of SRAM-based FPGAs (e.g., Atmel rad-hard FPGAs [ATF280E 2007] or Lattice products [Lattice Semiconductor Corporation 2010]) that partially protect the device without forgoing reconfigurability. However, such devices, besides being characterized by high cost and high time-to-market, present low density and operational frequency, and are often supported by a not so stable toolchain [ATF280E 2007]. Therefore, we refer to stan-dard, not-hardened SRAM-based FPGAs, and we propose to apply mitigation techniques to make them suitable for their adoption in space applications, where reliability is a strict requirement due to the harsh environmental conditions and the expected long system' lifetime.

In fact, the widespread diffusion of this kind of devices has led to the investigation and definition of design techniques and methodologies for hardening FPGA-based systems, with the aim of exploiting such systems in mission-critical scenarios. In general, hardening techniques are mostly based on spatial redundancy, whereas recovery, self-healing techniques exploit the FPGAs' (partial) reconfiguration capability, allowing to cope with the occurrence of faults by re-programming the faulty parts (e.g., [Carmichael et al. 2000; Samudrala et al. 2004; Bolchini et al. 2011a]).

Indeed, as hardened systems require many resources due to their size and complexity, a single FPGA may not suffice in terms of available resources, especially if the initial system uses most of the fabric resources (such as, for example, when implementing a configurable software-defined radio system), and multi-FPGA become an attractive solution to host hardened reconfigurable systems. Moreover, the multi-FPGA choice allows for a more flexible implementation of systems of a relevant size that can hardly fit on a single FPGA in their nominal, not hardened version. This situation either requires bigger (and more expensive) hardened FPGAs, or multiple standard FPGAs to host a hardened version of the system. In the latter case, the additional amount of resources allow not only to tolerate transient faults, but also to mitigate the effects of permanent ones, for which hardened FPGAs have no inherent solution. As a result, the adoption of multi-FPGA platforms allows for a more flexible solution against both transient and permanent faults.

While fault detection and masking techniques have been widely addressed in the case of systems based on a single FPGA (e.g., [Samudrala et al. 2004; Morgan 2006; Bolchini et al. 2011a]), the problem extended to multi-FPGA platforms has been rarely taken into account. The multi-FPGA scenario raises several issues, such as the partitioning of the system among the available devices, and the approaches suitable for the single FPGA scenario cannot be directly adopted. In the literature, no complete design methodology handling all the peculiar issues of our example scenario has been proposed yet, a gap we aim at filling with our work.

The paper proposes a reliability-aware methodology for designing embedded systems on multi-FPGA platforms for mission-critical applications, with the aim of exploiting standard, not-hardened SRAM-based FPGAs. The objective has been supported by the European Space Agency, as described in [Fossati and Ilstad 2011], where the vision about the future of embedded systems in space applications has been presented. The idea behind this work is to design a self-healing system by exploiting fault masking techniques in association with the partial dynamic reconfiguration. Two categories of faults are identified, based on their impact on the device elements, such that physical damage occurs or not; i) *recoverable* faults, transient problems that can be fixed without causing a lasting effect, and ii) *non-recoverable* faults, those that cause a permanent problem, making the affected portion of the fabric unusable. Recoverable faults can be mitigated by reconfiguring the system (and possibly only the faulty sub-system portion) with the same configuration used before fault occurrence, whereas non-recoverable faults, being characterized by a destructive effect, lead to the necessity of relocating the functionality to a non-faulty region of the device. The implemented system must be able to autonomously detect the occurrence of faults, and to mitigate their effects, thus, the methodology here proposed allows the overall system to continue working even if faults occur, increasing both system's reliability and lifetime.

The problem has been tackled by following a bottom-up approach. While some of our previous work dealt with specific activities (e.g., partitioning, floorplanning), we here present a complete design methodology. The main innovative contributions are summarized as follows:

— *Definition of an overall reliable multi-FPGA system with replicated control architecture*. Rather than making each FPGA an independent fault tolerant sub-system, able to locally detect and recover from faults, we have envisioned a solution where each FPGA is in charge of monitoring and, in case of faults, reconfiguring another device of the multi-FPGA system. The aim is to achieve a higher level of reliability in the overall system, trying to avoid the single point of failure characterizing the centralized solution and requiring to be implemented onto a particular device (e.g., an ASIC or an antifuse-based FPGA).

— *Design of a control system in charge of implementing the suitable recovery strategy based on the type of fault*. We propose the design of the controller in charge of managing the fault recovery of the multi-FPGA platform, contributing to the creation of the reliable system. More precisely, we introduce a reliability-aware Reconfiguration Controller, aimed at performing the envisioned fault classification and managing the reconfiguration process of the faulty parts of the architecture to mitigate fault effects. We present a replicated control solution, that avoids possible single points of failures.

— *Definition of a complete flow for designing autonomous fault tolerant systems on multi-FPGA platforms*. In the literature, methodologies for designing autonomous fault tolerant systems on multi-FPGA platforms have not been proposed yet to our knowledge. We aim to fill this gap by introducing a complete design flow for such systems. Furthermore, we describe the developed prototype framework, that implements the flow and automates, as much as possible, the design, hardening, and implementation of the envisioned system.

This article is structured as follows: Section 2 presents the related work and Section 3 provides background. Section 4 presents the proposed reliability-aware design methodology. Section 5 describes the proposed design flow and Section 6 focuses on the circuit hardening task. Section 7 describes the reliable controller. Section 8 discusses the methodology evaluation, performed by implementing a real case study. Finally, Section 9 presents the conclusions.

2. RELATED WORK

Our research proposes a methodology for designing reliable embedded systems on multi-FPGA platforms, with the final objective of increasing the system's reliability and lifetime in mission-critical scenarios.

In the past, much effort has been devoted to the study of reconfigurable systems and FPGAs have been widely investigated, as reported in [Hauck and DeHon 2007]; indeed today the topic constitutes an active research field, receiving a lot of attention with respect to dependability issues for FPGA-based solutions. In this wide scenario, we lean on some state-of-the-art techniques about reconfigurability and we focus on the issues related to reliability and multi-FPGA platforms.

While fault detection and masking techniques have been widely addressed in the case of systems based on a single FPGA (e.g., [Carmichael et al. 2000; Emmert et al. 2000; Samudrala et al. 2004; Morgan 2006; Bolchini and Miele 2008; Bolchini et al. 2011a]), the problem extended to multi-FPGA platforms has been rarely taken into account. Moreover, only faults recoverable by device (partial) reconfiguration are usually targeted ([Carmichael et al. 2000; Morgan 2006; Bolchini and Miele 2008]), whereas faults physically damaging the device are seldom analyzed. The approach in [Bolchini et al. 2011a] presents a design flow for implementing digital systems on single SRAM-based FPGAs with soft error mitigation properties. While it could be adapted to multi-FPGA platforms and extended to faults with destructive effect, we claim that an ad-hoc solution for this scenario would consistently improve the system's performance and its reliability. A comparison between the approach proposed in [Bolchini et al. 2011a] and ours is reported in Section 8, supporting the motivations at the basis of the proposed design flow.

In literature, when considering multi-FPGA platforms, the available devices are usually exploited to host replicas of the main system, as in [Smith and de la Torre 2006], where three FPGAs are used to apply the classical Triple Modular Redundancy (TMR) technique on the whole circuit. Each FPGA hosts the same configuration and an external radiation-hardened ASIC implements a controller, that acts as a TMR voter. Indeed, this is a different scenario with respect to the one we envision; we aim at a more flexible exploitation of the devices, providing a scalable solution, independent of the number of used FPGAs. In addition, in the mentioned work the controller needs to be implemented on particular ASIC technology, eliminating most of the advantages of having SRAM-based FPGAs in the system, namely flexibility and relatively low cost.

To conclude, no complete design methodology handling all the peculiar issues of the considered scenario has been proposed yet, a gap we aim at filling with our work.

3. WORKING SCENARIO

This section introduces the background elements useful to understand the rest of the paper, namely the adopted fault model and the proposed fault classification for the selected space environment.

3.1. Fault model

When designing fault mitigation strategies, the *single fault assumption* is traditionally considered. It implies that i) faults occur one at a time and ii) the time between the occurrence of two subsequent faults is long enough to allow the detection of the first fault before the second one occurs. This assumption is adopted at a high abstraction level, by considering faults affecting independent portions of the system; in our approach, hardening techniques are applied at a functional level such that they work also for multiple bit-flips, provided they occur in independent areas, as discussed in Section 4. It is worth noting that the single fault assumption is usually introduced to avoid faults' accumulation, possibly mining the effectiveness of fault tolerance techniques due to masking/biasing effects, a realistic assumption for the considered scenario.

Indeed, the fault only produces an observable effect (i.e., a failure) if i) the fault occurs in a used resource and ii) the applied input (sequence) is such that a difference in the data/behavior is caused with respect to the fault-free situation. Thus, since it may happen that not all available resources are actually used, the observability of a fault is related to the probability of the fault to hit a used resource. Furthermore, it is necessary that the adopted detection mechanism identifies the fault situation with short latency between the fault occurrence and the failure.

3.2. Fault classification

When considering SRAM-based FPGAs, it is possible to identify two types of memory that can be subject to faults; the one storing the application data being processed and the one storing the configuration bitstream, that defines the functionality performed by the reconfigurable fabric. If the data memory is corrupted, an erroneous value is produced, whereas, when a fault corrupts a configuration memory element, it modifies the programmed functionality. Configuration memory accounts to more than 95% of the fraction of the device sensitive to faults [Morgan 2006], thus an erroneous functionality is the most common produced effect. Nevertheless, we aim at covering faults affecting both kinds of memories. To cope with erroneous values, a reset of the application can be performed or feedback loops to propagate the correct values to the registers can be added (in an X-TMR fashion [Xilinx Inc. 2006]), whereas, to restore erroneous functionalities, the configuration must be re-programmed.

We identify two categories of faults based on the possibility to recover from them by reconfiguration:

— *Recoverable faults*, that can be mitigated by reconfiguring the system, and possibly only the faulty sub-system portion through partial reconfiguration, with the same configuration used before fault occurrence, and
— *Non-recoverable faults*, that permanently compromise part or all of the device, such that further use of the corrupted portion of the device must be avoided and the logic hosted must be moved in a different location.

The identification of the type of fault occurred on the device is thus fundamental to apply the suitable recovery strategy.

In the following, we introduce the main faults affecting FPGAs in the space environment and we distinguish them in the two categories.

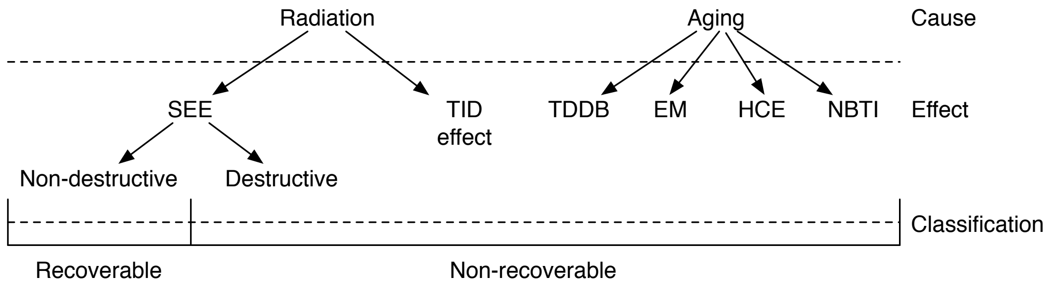

Fig. 1. Cause-based fault classification in the space environment.

3.3. Space environment

The space environment has been selected as working scenario for the stringent reliability requirements of space applications, due to the harsh environmental conditions and the difficulty of system maintenance. Two problems have been identified as main causes of faults in space; radiations and device aging. In the following, we take into account and extend the fault characterization proposed in [Bolchini and Sandionigi 2010].

Radiations are the most common cause of faults in space. The effects can be divided into two main groups [ECSS-E-ST-10-12C 2008]: i) *Single Event Effect* (SEE), that is a measurable effect resulting from the deposition of energy from a single ionizing particle strike, and ii) *Total Ionizing Dose* (TID) effect, that is a cumulative long term ionizing damage. SEEs are distinguished into non-destructive or destructive, based on their effect; non-destructive SEEs can be recovered by resetting or reconfiguring the device, whereas destructive SEEs have a persistent effect. Non-destructive SEEs are commonly called Single Event Upsets (SEUs), modeled as bit flips. Particular types of SEUs are Single Event Disturbs (SEDs), momentary voltage excursions at nodes, and Single Event Functional Interrupts (SEFIs), interrupts leading to temporary non-functionality of the affected device. SEDs are treated as general SEUs, whereas SEFIs are not taken into account in this paper since they are negligible in occurrence [Carmichael and Tsen 2009]. Destructive SEEs are Single Event Latchups (SELs), energy from a charged particle leading to an excessive supply power. They are not taken into account in this work as test reports on SRAM-based FPGAs reveal that no SEL was observed during the experiments, up to the maximum tested Linear Energy Transfer of tens of $MeV \cdot cm^2/mg$ [Poivey et al. 2007; Petrick et al. 2004], that is sufficient for the considered applications. Finally, also TID effects, defining the total sum of radiations hitting the FPGA, have the potential to destroy the device.

Device aging is the other analyzed cause, an important aspect for long-lasting space missions, where system maintenance or substitution is difficult. It causes the following effects with destructive outcomes [Srinivasan et al. 2006; Srinivasan et al. 2008]: i) *Time Dependent Dielectric Breakdown* (TDDB), that is a breakdown caused by charge flow through the oxide, ii) *Electromigration* (EM), that is a development of voids in metal lines due to heavy current densities over a period of time, iii) *Hot-Carrier Effects* (HCE), that are interface traps affecting the I-V characteristics of the transistors, and iv) *Negative Bias Temperature Instability* (NBTI), that is the degradation dependent on the time a PMOS transistor is stressed in the circuit. In the field of FPGAs, these faults are just being recently addressed, also due to their rareness.

To summarize, recoverable faults are the ones caused by radiations without a destructive effect, whereas non-recoverable faults are those caused by radiations with destructive effect, radiations' accumulation, and device aging. Figure 1 shows such fault classification based on the analyzed causes and effects.

4. DESIGN METHODOLOGY

The proposed reliability-aware methodology allows the design of self-healing systems implemented on multi-FPGA platforms, with the final objective of exploiting commercial SRAM-based FPGAs for mission-critical applications. While some aspects can be taken from previous solutions available in literature, several open issues for the proposed methodology exist and are here addressed. As shown in Figure 2, the methodology is composed of the following main activities, each of them tackling one of the identified relevant issues: i) architecture definition, ii) design space exploration, and iii) fault management definition.

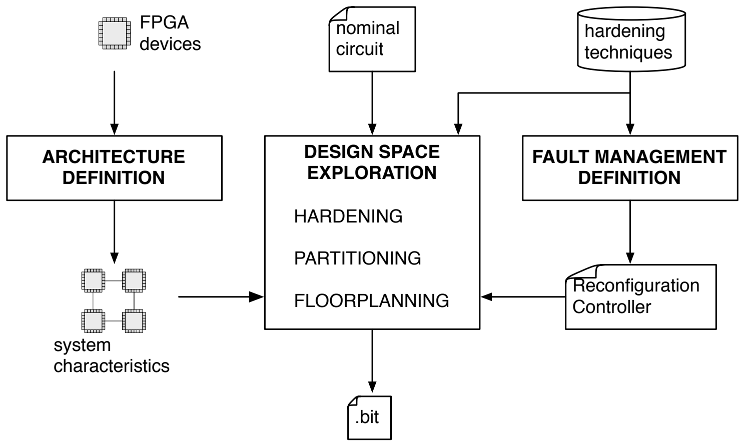

Fig. 2. Proposed reliability-aware design methodology.

4.1. Architecture definition

The first input to be provided is the architecture at the basis of the self-healing system, in terms of the number of available SRAM-based FPGAs and their communication infrastructure, for both data exchange and configuration control. In particular, we adopt a solution where the reconfiguration controller in charge of triggering a recovery action upon fault detection is distributed onto the various FPGAs, as preliminarily presented in [Bolchini et al. 2010]. As a result, the system is hosted on a platform with multiple SRAM-based FPGAs connected to each other in a mesh topology, as shown in Figure 3.

Each FPGA hosts

— a hardened portion of the entire circuit, organized in *independently recoverable areas* (IRAs [Bolchini et al. 2011a]) that autonomously detect, mask/tolerate, and signal the occurrence of a fault, and
— a *Reconfiguration Controller*, that is the engine in charge of monitoring the error signals to trigger, when needed, the reconfiguration of the faulty part of the circuit.

Fault detection and tolerance techniques, e.g., by means of spatial redundancy, are used to identify the occurrence of the fault and to localize the corrupted portion of the FPGA, the independently recoverable area. We envision the circuit to be partitioned into areas, each one reconfigurable independently by the remaining part of the device upon fault occurrence. We assume a uniform distribution of radiations (and consequent faults) on the FPGA fabric, therefore to avoid having areas more susceptible than others, the methodology will have as an objective the definition of areas similar in terms of size and used resources. Furthermore, since the possibility of a fault to cause an observable

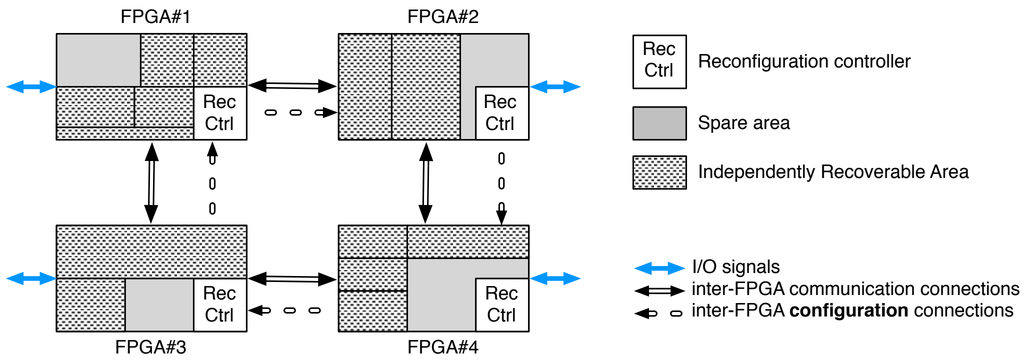
FPGA#1
FPGA#2
FPGA#3
FPGA#4
Rec Ctrl
Rec Ctrl
Rec Ctrl
Rec Ctrl
Rec Ctrl
Reconfiguration controller
Spare area
Independently Recoverable Area
I/O signals
inter-FPGA communication connections
inter-FPGA **configuration** connections

Fig. 3. Proposed system architecture.

error (and thus trigger a recovery action) depends on its controllability and observability with respect to the implementation of the logic onto the FPGA resources, we also assume such a fault-error relation to be homogeneous across the entire circuit/device.

Each IRA generates error signals and they are all collected and monitored by a Reconfiguration Controller (`Rec Ctrl` in Figure 3) in charge of triggering a reconfiguration as soon as an error is observed. More precisely, the proposed approach defines an association between the IRAs hosted on an FPGA and a Reconfiguration Controller in charge of managing their reconfiguration hosted on a different FPGA. The methodology does not perform the association directly, an operation that is left to the designer, who must only guarantee that the available I/O pins on the various boards cover the requirements. Altogether, the only constraints are i) that each FPGA is monitored by a single neighbor FPGA, ad ii) that the number of I/Os suffices. The controller, adopts a classification strategy to determine the nature of the effects of the type of fault (recoverable or not) and performs the appropriate recovery action. Should the fault be considered recoverable, the bitstream portion related to the faulty area is reloaded, otherwise the faulty area is relocated to a spare region reserved for this purpose. Before performing relocation to a spare area, the region is checked to verify the absence of degradation effects by means of structural tests (e.g., [Renovell 2002], [Armin et al. 2006]).

Reliability requirements are expressed for the Reconfiguration Controller also, in case a fault occurs in the area hosting it, and such that no undesired and/or erroneous reconfigurations take place. Therefore, the module has been designed to be able to autonomously detect the occurrence of a fault and to mitigate its effects while waiting for recovery handled by another controller hosted onto another FPGA. Rather than making each FPGA an independent fault tolerant sub-system, able to locally detect and recover from faults, we have envisioned a solution where the Reconfiguration Controller hosted on an FPGA is in charge of reconfiguring the areas hosted on a neighbor FPGA. The aim is to achieve a higher level of reliability in the overall system, trying to avoid the single point-of-failure characterizing the centralized solution and requiring to be implemented onto a particular device (e.g., an ASIC or an antifuse FPGA, as in [Smith and de la Torre 2006]).

4.2. Design space exploration

The design space exploration is the core of the proposed methodology; this activity is devoted to the hardening of the nominal circuit and its distribution on the multi-FPGA platform. As the name suggests, we are not just pursuing *a* working solution, rather the approach identifies the (set of) solution offering the most convenient trade-off with respect to the designer's selected metrics.

The activity is composed of three main steps (discussed in details in Section 5), each one performing an exploration of the possible alternatives: i) *hardening*, that selects and applies fault detection/tolerance techniques, ii) *partitioning*, that distributes the obtained reliable circuit among the available FPGAs, and iii) *floorplanning*, that positions each sub-circuit within the related device.

4.3. Fault management definition

The fault management definition activity (detailed in Section 7) identifies how to classify the faults and cope with their occurrence. Classification is used to distinguish recoverable and non-recoverable faults, as the recovery action differs in the two situations. Furthermore, it is necessary to define the engine in charge of implementing the recovery strategy. The controller must be able to monitor a parametric number of error signals and, based on the identified type of fault, it must perform the suitable recovery action.

5. DESIGN FLOW

The definition of an effective design flow for the implementation of a hardened system onto the multi-FPGA platform has taken into account various alternatives to cope with the complexity of carrying out the design space exploration for the different tasks, namely hardening, partitioning and floorplanning. The result is the design flow reported in Figure 4.

The methodology considers a module-based design approach, commonly adopted for FPGAs. Hence, the model of the circuit is specified in terms of structural description, constituted of components interconnected by wires. For each one of the components, implementation costs (e.g., necessary resources) are required and they can be either extracted by means of a preliminary synthesis with commercial tools, if the system source specification is available, or directly provided by the designer, in case of a black-box approach to protect IPs. Moreover, a parsing task produces a more agile, graph-based representation of the system, used by the other tasks of the flow, together with the information on the components' cost and the platform model (e.g., available resources and distribution on the reconfigurable fabric).

The flow consists of the following tasks: i) preliminary partitioning, ii) circuit hardening, iii) solution refinement, and iv) floorplanning. When a feasible solution is identified, the implementation phase integrates the Reconfiguration Controllers, one for each FPGA, with the hardened circuit, building the proposed overall self-healing system.

5.1. Preliminary partitioning

The preliminary partitioning task distributes the nominal circuit's components among the FPGAs. The goal is to identify an initial uniform distribution of the circuit, also minimizing inter-FPGA communication. At this point, the number of top-level components is usually limited, allowing for the definition of a partitioning problem of acceptable complexity in pursuit of a uniform distribution. Do note that the application of space redundancy-based hardening techniques has an impact in terms of additional components (replicas and comparators/voters) and connections, leading to complex systems (in the order of tens of components) for which it might be time-consuming to find a solution. Therefore, this preliminary partitioning identifies for each component its mapping onto one of the FPGAs; the hardening phase will then introduce the necessary replicas and related components.

5.2. Circuit hardening

The reliability requirement for the overall system is fault tolerance, therefore the circuit hardening task applies fault mitigation techniques to define the so-called *independently recoverable areas* (IRAs), that are portions of the FPGA that monitor and mask the occurrence of faults and that can be (partially) reconfigured if necessary. It is beyond the scope of this work to investigate the most convenient hardening solution for the component implemented on the FPGA (e.g., there could be a robust library version of a module). The only requirement the methodology introduces, it that the hardened implementation defines an IRA that detects faults and masks their effects, generating the necessary error signals; in this paper we adopt Triple Modular Redundancy (TMR) within the IRA. Based on these error signals, dynamic partial reconfiguration can be triggered to recover from the fault.

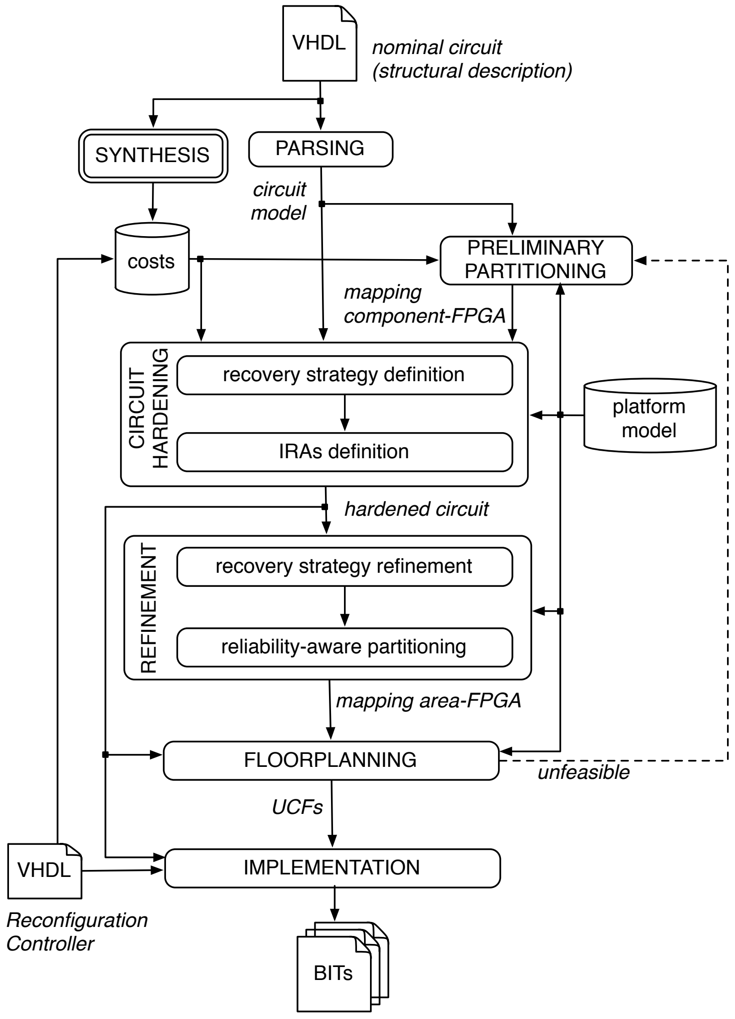

Fig. 4. Proposed design flow.

The hardening task of the flow (an extension of the approach presented in [Bolchini et al. 2011a] adapted to the multi-FPGA scenario) evaluates the possibility to apply TMR to the single component or to groups of components, optimising the number of voters to be introduced to limit overheads and connections. Moreover, all three replicas and the voter can be hosted in the same IRA or they can be distributed on different IRAs to reduce the portion of FPGA to be reconfigured in case a fault is detected. Indeed, the higher the number of areas the finer the fault detection and recovery granularity, but, at the same time, a memory with higher capacity to store the recovery bitstreams is necessary. As a consequence, a design space exploration, described in the following section, analyzes the trade-off in terms of costs (e.g., FPGA resources, bitstream size and number) and benefits (e.g., number of tolerable faults).

5.3. Solution refinement

This task aims at improving the obtained reliable solution, considering that the initial partitioning of the circuit has been performed prior to it hardening and that, possibly, the spare area reserved on the FPGAs allows for more faults to be tolerated with respect to the initial estimation.

5.3.1. Recovery strategy refinement. The previous task has defined the IRAs hosted on each FPGA, starting from an estimation of the number of non-recoverable faults to be tolerated on each device ($\#faults$). The recovery strategy refinement activity re-evaluates $\#faults$, with respect to the the hardened circuit.

For each FPGA device d, the maximum number of tolerated non-recoverable faults (max_faults_d) is computed by considering the number of times the largest IRA hosted on the device can be relocated in the spare region. The other contribution is given by the available memory capacity with respect to the bitstream size; thus, $\#faults$ is computed as:

$$\#faults = \min(\#faults_d \mid \sum_{i=0}^{\#faults_d} \#areas_d^i \leq \frac{memory\_size}{bitstream\_size}, \forall d \in D)$$

where $\#areas_d$ is the number of IRAs hosted on FPGA d and $\#faults_d \leq max\_faults_d$ is the number of tolerated faults.

5.3.2. Reliability-aware partitioning. The reliability-aware partitioning task re-distributes the independently recoverable areas among the available FPGAs, by taking into account also possible recovery actions needed during the system's lifetime ([Bolchini and Sandionigi 2011]). It reserves a suitable spare region for relocations. The output of this task is a mapping, specifying for each area its hosting on one of the devices.

5.4. Floorplanning

The floorplanning task finds a suitable placement of the hardened areas on the FPGA, including the controller. Besides defining the initial placement, the complete set of recovery actions must be considered (see Figure 6).

The process is automated by means of a resource-aware floorplanner supporting 2D reconfiguration constraints, necessary to enable partial scrubbing. The adopted floorplanner stems from classical approaches, using an enriched sequence pair representation and exploiting the simulated annealing engine, customized to cope with the specific scenario.

This activity borrows from a reconfiguration-aware floorplanner presented in [Bolchini et al. 2011b] and extends it to take into consideration the multi-FPGA scenario, as well as the constraints on the regions to be avoided. An initial placement is generated, then, for each possible single non-recoverable fault, another placement is found by avoiding the faulty region. The process continues by following the order of the recovery actions from non-recoverable faults, as shown in Figure 6.

The floorplanning task produces the placement constraints for each area on the related FPGA. The hardened circuit is placed on the multi-FPGA platform and, by integrating the necessary Reconfiguration Controllers (one for each FPGA), the proposed reliable system is implemented.

If the floorplanning task reveals that the solution is unfeasible, the flow is repeated starting from the preliminary partitioning task. To avoid an infinite loop, the designer can intervene and add constraints to help the solver in finding a new solution, thus converging. In practice, there are usually feasible mappings unless the selected number of FPGAs for implementing the self-healing system is too limited (or eventually close to the limit). To this end, a new design flow has also been defined, discussed in the next subsection, to support the designer in the exploration of the size of the multi-FPGA architecture able to tolerate a given number of faults.

5.5. Design flow for surviving a fixed number of non-recoverable faults

The design flow adopted up to this point takes as input the multi-FPGA platform, as specified by the designer. As an alternative, it is also possible to “reverse” the design flow, such that the designer specifies the number of non-recoverable faults to be tolerated based on the desired system lifetime; in this scenario, the methodology identifies a suitable multi-FPGA architecture able to achieve such goal.

The alternative design flow is shown in Figure 5; it requires the model of the single FPGAs and the number of non-recoverable faults to be tolerated, to identify the most convenient architecture (minimising the number of necessary FPGAs) able to achieve such goal. The circuit components are taken in input by the reliability-aware iterative partitioning task, that distributes them among the necessary FPGAs. A hardened version of the circuit is taken into account, where TMR is applied on each component. By starting from two FPGAs, the partitioner distributes the hardened components among the devices and increases the number of required FPGAs until it identifies the suitable amount of spare regions for tolerating the non-recoverable faults.

The rest of the flow, at this point, proceeds by carrying out the same activities of the other flow, in order to define the implementation of the self-healing system onto the identified architecture.

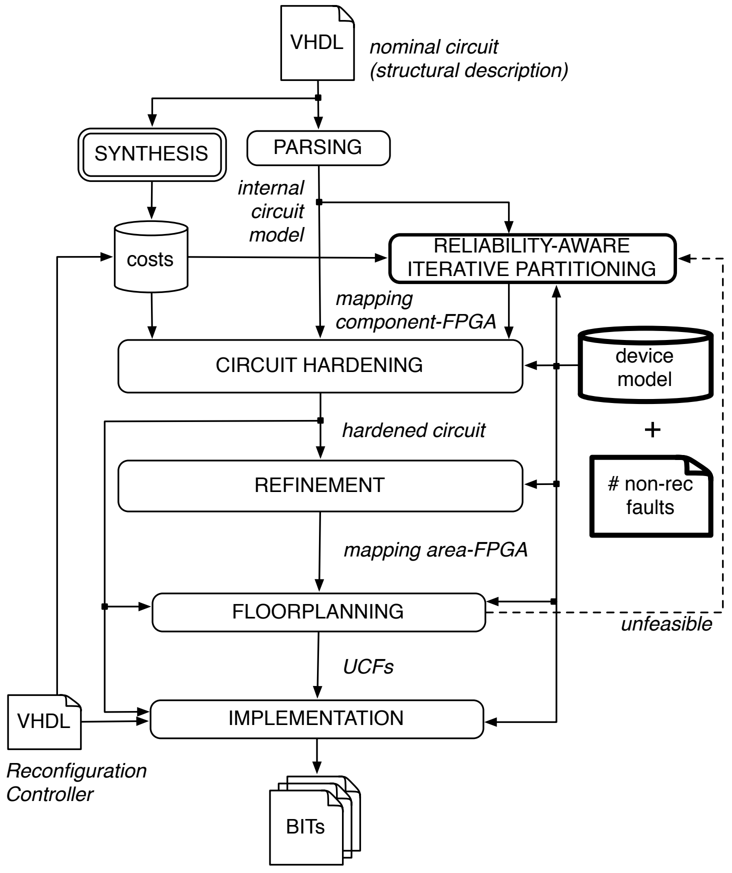

Fig. 5. Flow for fixed number of non-recoverable faults.

6. CIRCUIT HARDENING

The circuit hardening task performs a design space exploration composed of two main activities: Definition of the recovery strategy and definition of the IRAs.

6.1. Recovery strategy definition

The first activity performed when hardening the circuit defines the parameters constraining the definition of the independently recoverable areas on the FPGAs. Such parameters are related to the adopted recovery strategy, described in the following.

When reconfiguring to cope with non-recoverable faults, a new configuration that replaces the faulty region in the FPGA with a spare one is loaded. The strategies for defining the recovery configurations can be classified into two categories:

— *On-line bitstream computation*, that dynamically creates the recovery configurations, as in [Montminy et al. 2007], and
— *Off-line bitstream computation*, that creates the alternative configurations during the design phase, as in [Mitra et al. 2004].

The former strategy means that only one bitstream is saved in memory and that, if needed, it is modified at runtime to assign the functionalities to the correct areas. Partial dynamic reconfiguration and bitstream relocation provided by Xilinx FPGAs are used for defining on-line approaches. These approaches do not address the reliability issues related to the system's static region defined by the Xilinx design flow, that contains the global communication infrastructure and, possibly, the internal Reconfiguration Controller. Actually, a fault affecting the static region cannot be recovered and would affect the overall system. Moreover, when an independently recoverable area is moved to a new location, a new floorplanning is often necessary to obtain a feasible implementation because of new interconnections and constraints. Finally, the strategy would make the Reconfiguration Controller heavily dependent on the actual FPGA and it would consistently increase its complexity. For these reasons we have not considered on-line bitstream computation in our approach.

However, the other strategy, off-line bitstream computation, requires the storage of all the complete bitstreams that might be needed during the system's lifetime, thus a large memory is required. In fact, the approach consists of computing all the bitstreams that might be needed before deploying the system and, at runtime, only the correct bitstream is loaded. For each sequence of occurred non-recoverable faults and for each independently recoverable area the circuit is divided into, a bitstream must be stored in memory (refer to Figure 6). Indeed, it is necessary to maximize i) the number of non-recoverable faults the system can autonomously recover from without any external action, to improve the system's lifetime, and ii) the number of areas, both to allow fault detection at finer granularity and to reduce the average partial scrubbing time required for coping with recoverable faults. Also, it is required not to exceed the capacity of the bitstream memory.

Thus, the following parameters related to the relocation strategy are defined:

— $\#faults$, that is the number of non-recoverable faults for each FPGA the system can autonomously cope with, and
— max_areas, that is the maximum number of independently recoverable areas the sub-circuits on the FPGAs are divided into.

To compute $\#faults$, a hardened version of the circuit is taken into account, partitioning each component in a different area and applying TMR. Then, the maximum number of tolerated non-recoverable faults for each device d (max_faults_d) is the number of times the greatest area hosted on the device can be moved onto a not used, fault-free region. $\#faults$ is the minimum value among the identified numbers of tolerated non-recoverable faults:

$$\#faults = \min\left(max\_faults_d, \forall d \in D\right)$$

When considering multiple subsequent non-recoverable faults, a specific bitstream is required for each sequence of occurred faults. Therefore, assuming a given number of independently recoverable

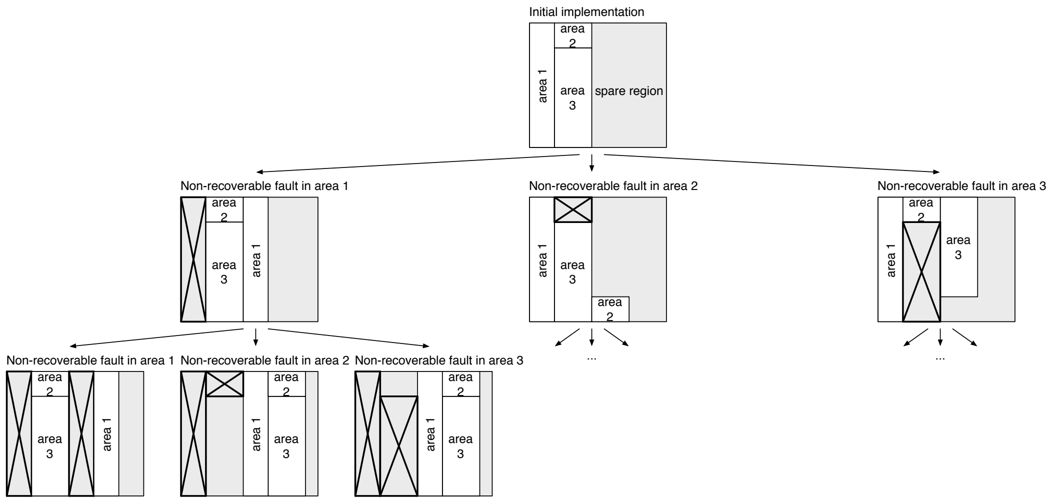
Initial implementation
area 1
area 2
area 3
spare region
Non-recoverable fault in area 1
Non-recoverable fault in area 2
Non-recoverable fault in area 3
...

Fig. 6. The adopted recovery strategy for non-recoverable faults.

areas $\#areas_d$, equal to the number of components hosted on FPGA d, to tolerate `#faults` the following number of alternative bitstreams for each device has to be generated:

$$\#bitstreams_d = \sum_{i=0}^{\#faults} \#areas_d^i, \forall d \in D$$

It is worth noting that $\#bitstreams_d$ can be very high also for low values of $\#faults$, hence a very large memory should be required. For example, considering 15 independently recoverable areas and 3 non-recoverable faults, 3616 bitstreams must be generated; since the dimension of a bitstream is typically of the order of MB, a memory of some GB is required. This is a large memory in the embedded systems scenario, especially when it must be protected for critical scenarios.

Thus, the number of areas `max_areas` is set by fulfilling the memory constraint:

$$max\_areas = \max(\#areas \mid \sum_{i=0}^{\#faults} \#areas^i \leq \frac{memory\_size}{bitstream\_size})$$

being $memory\_size$ and $bitstream\_size$ the size of the memory and of the bitstream for the selected device, respectively. max_areas is considered when defining the independently recoverable areas, as described in the following.

6.2. Independently recoverable areas definition

By considering the parameter max_areas defined in the previous activity, the independently recoverable areas composing the hardened circuit are defined. For each FPGA on the platform, the components of the sub-circuit hosted on the device are grouped and a hardening technique is applied to each identified group. We select TMR for fault tolerance and, borrowing from [Bolchini et al. 2011a], we envision different strategies of applying it based on the distribution of the replicas. The interested reader can refer to the original work for the details. Each of these strategies is a hardening technique, that identifies the independently recoverable areas where the groups of components are mapped; in general, other hardening techniques to be taken into account together with these ones, or some of them can be disabled, to customise the set of strategies for obtaining the IRAs.

Although referring to the work in [Bolchini et al. 2011a], a new formulation has beed developed to fit this multi-FPGA design flow. The characterising elements are the following ones. The circuit is modeled as a set of components, $c \in C$, interconnected with each other by wires; $comp\_wires_{c_1,c_2}$ is the number of wires between components c_1 and c_2, and $comp\_out_c$ is the number of output wires of component c. Each component is characterized by its resource requirements, $comp\_res_{c,r}$, that is the number of resources of type r required by component c. The activity partitions the nominal circuit in groups, $g \in G$, where G is the set of groups, and applies a hardening technique, $t \in T$, where T is the set of techniques, to each of them. Each group is mapped on one or more areas, $a \in A$, where A is the set of areas. Thus, each area of the hardened circuit hosts one or more components of the nominal circuit and constitutes an independently recoverable area. The parameters used in the circuit hardening are reported in Table I.

Table I. Parameters used in the circuit hardening task.

Parameter	Description
C	Set of components
G	Set of groups
T	Set of hardening techniques
A	Set of areas
R	Set of resources
$comp\_res_{c,r}$	Number of resources of type r required by component c
$comp\_wires_{c_1,c_2}$	Number of wires between components c_1 and c_2
$comp\_out_c$	Number of output wires of component c

The definition of the IRAs hosted on each device has been modeled as a mixed-integer linear programming (MILP) problem, whose inputs are the nominal circuit (in terms of components and required resources) and the parameter max_areas identified by the previous activity. A different approach to the design space exploration for the identification of the most convenient solution in the partitioning of the system onto a single FPGA has been proposed in [Bolchini and Sandionigi 2011], by means of a two-levels engine based on genetic algorithms. Here, we adopt the MILP formulation to identify the most convenient solution, presenting a comparison with the one obtained from the other approach in Section 8.3.

The output of the elaboration is *for each component c, which group g it belongs to, and what hardening techniques t will be applied*. In terms of the specific model, this corresponds to a decision variables $x_{c,g,t}$ that equals to 1 if component c belongs to group g where technique t is applied, 0 otherwise.

Indeed, to account for the number of identified groups and the communication between groups, the following binary variables are introduced; $y_{g,t}$, that equals to 1 if group g where technique t is applied hosts at least a component, and $z_{c_1 c_2,g,t}$, that equals to 1 if components c_1 and c_2 are assigned to group g where technique t is applied.

The design space exploration will evaluate the possible alternatives of assigning component c to group g with technique t, by comparing them with respect to the following metrics, here adopted:

— *Distribution uniformity*, that is generally desirable to divide the circuit into portions of roughly equal sizes, and is even more important when reliability is taken into account, to have balanced areas reserved for relocations.
— *Number of areas*, that must be maximized both to allow fault detection at finer granularity and to reduce the average partial scrubbing time required for coping with recoverable faults.
— *Number of wires between groups*, that must be minimized for reducing the resource requirement of the necessary TMR voters.

A set of constraints need to be added to the specification of the MILP problem, deriving from the rules associated with identifying for each component a group, a hardening technique and the necessary amount of resources (wires, LUTs, . . .), listed in in Table II and briefly discussed in the following, by providing for each constraint the informal meaning.

Groups definition constraints. The first constrain guarantees that each component must be assigned to a group and this is imposed by stating that the sum of all assignments $z_{c_1,c_2,g,t}$ with respect to groups and techniques must equal 1. Constraints C2 and C3 define the valid values of the variables $y_{g,t}$, by identifying the maximum and the minimum bounds, respectively. Moreover, and each group with at least a component must have a technique applied to it, and Constraint C4 imposes it by using variables $y_{g,t}$.

Constraint C5 computes the number of allocated areas by taking into account the number of groups and the number of areas required by the applied technique ($\#areas\_req_t$). Finally, the number of areas allocated for mapping the groups cannot be higher than max_areas, required by the relocation strategy, or lower than min_areas, as defined by Constraints C6 and C7, respectively.

Wires between groups constraints. The number of connections between groups is identified by computing the number of wires between pairs of components hosted on different groups, c_1 and c_2, as defined by Constraint C11, that refers to variables $z_{c_1,c_2,g,t}$. Constraints C8, C9 and C10 define the valid values of the variables $z_{c_1,c_2,g,t}$, by identifying the maximum bound (Constraints C8 and C9) and the minimum one (Constraint C10).

Resources distribution constraints. The resource requirement for each group of components to be hosted on an IRA is computed by considering the amount of resources required to implement the components and the voter; this is specified in Constraint C12. Because this amount depends also on the replicas introduced by the hardening technique, parameters $w\_comp_{t,a,r}$ are defined to account also for this contribution, while parameter $w\_voter_{t,a,r}$ defines the multiplication factor for wires (based on the number of voted wires). Indeed, the resource requirements of the areas cannot exceed the total amount of available resources on the FPGA, as defined by Constraint C13. When dedicated/specialized resources (e.g., BRAMs, DSPs) are not sufficient, it is possible to “convert” them in terms of slices-equivalent; such conversion is made for all the replicas of a module to have uniform areas. Constraints C14 and C15 identify the areas obtained using the minimum resource requirement and the maximum one, respectively, suitable to host the IRA. It is worth noting how the area with the minimum resource requirement is defined by considering only the areas hosting components; we exploit the parameter M, a big enough value to avoid wrong values of min_r when considering areas without components. Finally, we introduce Constraint C16 to drive the design space exploration towards solutions with a uniform distribution of components; parameter gap_r evaluates the gap between the largest area and the smallest one to be used in the minimization.

By indicating with:

$$\alpha = \sum_{r \in R} \frac{gap_r}{dev_r}$$

$$\beta = (1 - \frac{\#areas}{max\_areas})$$

$$\gamma = \frac{\sum_{g \in G} \sum_{t \in T} comp\_wires_{g,t}}{\sum_{c_1 \in C} \sum_{c_2 \in C} comp\_wires_{c_1,c_2} + \sum_{c \in C} comp\_out_c}$$

the adopted objective function is:

$$min(w_{res} \cdot \alpha + w_{areas} \cdot \beta + w_{wires} \cdot \gamma)$$

with

$$w_{res} + w_{areas} + w_{wires} = 1$$

where w_{res}, w_{areas}, and w_{wires} are designer-assigned weights.

7. RECONFIGURATION CONTROLLER

The main component responsible for the active self-healing process is the Reconfiguration Controller. It continuously monitors the error signals from the neighbor FPGA and, upon error de-

Table II. Independently recoverable areas definition constraints.

Groups definition	
C1	$\sum_{g \in G} \sum_{t \in T} x_{c,g,t} = 1, \forall c \in C$
C2	$y_{g,t} \leq \sum_{c \in C} x_{c,g,t}, \forall g \in G, t \in T$
C3	$y_{g,t} \geq x_{c,g,t}, \forall c \in C, g \in G, t \in T$
C4	$\sum_{t \in T} y_{g,t} \leq 1, \forall g \in G$
C5	$\#areas = \sum_{g \in G} \sum_{t \in T} y_{g,t} \cdot \#areas\_req_t$
C6	$\#areas \leq max\_areas$
C7	$\#areas \geq min\_areas$
Wires between groups	
C8	$z_{c_1,c_2,g,t} \leq x_{c_1,g,t}, \forall c_1 \in C, c_2 \in C, g \in G, t \in T$
C9	$z_{c_1,c_2,g,t} \leq x_{c_2,g,t}, \forall c_1 \in C, c_2 \in C, g \in G, t \in T$
C10	$z_{c_1,c_2,g,t} \geq x_{c_1,g,t} + x_{c_2,g,t} - 1, \forall c_1 \in C, c_2 \in C, g \in G, t \in T$
C11	$group\_wires_{g,t} = \sum_{c_1 \in C} \sum_{c_2 \in C} comp\_wires_{c_1,c_2} \cdot (x_{c_1,g,t} - z_{c_1,c_2,g,t})$
	$+ \sum_{c \in C} comp\_out_c \cdot x_{c,g,t}, \forall g \in G, t \in T$
Resources distribution	
C12	$group\_res_{g,t,a,r} = \sum_{c \in C} comp\_res_{c,r} \cdot x_{c,g,t} \cdot w\_comp_{t,a,r}$
	$+ group\_wires_{g,t} \cdot w\_voter_{t,a,r}, \forall g \in G, t \in T, a \in A, r \in R$
C13	$\sum_{g \in G} \sum_{t \in T} \sum_{a \in A} group\_res_{g,t,a,r} \leq dev_r, \forall r \in R$
C14	$min_r \leq M \cdot (1 - y_{g,t} \cdot req_{t,a}) + group\_res_{g,t,a,r}, \forall g \in G, t \in T, a \in A, r \in R$
C15	$max_r \geq group\_res_{g,t,a,r}, \forall g \in G, t \in T, a \in A, r \in R$
C16	$gap_r = max_r - min_r, \forall r \in R$

tection, triggers a reconfiguration based on the classification of the fault nature (recoverable vs. non-recoverable).

When an error is signaled from the monitored FPGA, the Fault Classifier identifies it as recoverable or non-recoverable, by implementing the algorithm proposed in [Bolchini and Sandionigi 2011]. The classification is based on the analysis of the fault's frequency. An area is considered affected by a non-recoverable fault when it has been "detected" as faulty during the last K subsequent observations.

Indeed, it is necessary to prevent erroneous reconfigurations by detecting also faults affecting the Reconfiguration Controller itself. We designed a low cost engine, relaxing the need to deploy it on ad-hoc device and we allowed scalable use of multiple FPGAs to build a reconfigurable reliable system. While the proposal in [Bolchini et al. 2010] achieves reliability by means of bitstream read-back, an expensive error-prone operation, here we introduce a novel hardened design. The controller provides autonomous fault detection capability and can be seen as an area that, in case of fault, is recovered or relocated.

In the following, the implementation details and the hardening of the Reconfiguration Controller are reported.

7.1. Implementation details

The structure of the reconfiguration controller is shown in Figure 7. The controller receives in input the `error_signals` generated by the monitored neighbour FPGA, and they are processed by the module implementing the fault classification algorithm. Furthermore, the module also receives an `enable` signal specifying when the error signals are to be considered valid (the Fault Classifier is disabled during the recovery phase). The areas' error signals are encoded with the two-rail code (TRC) whereas the enable signal, generated within the Reconfiguration Controller, is not encoded since faults in the overall controller are revealed by an error signal generated by the controller itself. Three are the Fault Classifier outputs: `fault_identified` to signal whether a fault has been detected, `fault_type` to specify whether it is recoverable or not, and `faulty_area` to identify the area where the fault occurred. This information is used to trigger the appropriate reconfiguration/relocation strategy, managed by the module in charge of executing the re-programming of the neighbor FPGA. The classification of the fault upon error detection additionally uses the `last_ira` information to specify the area identified as faulty the last time an error has been detected, together with the threshold K for classifying a fault as non-recoverable.

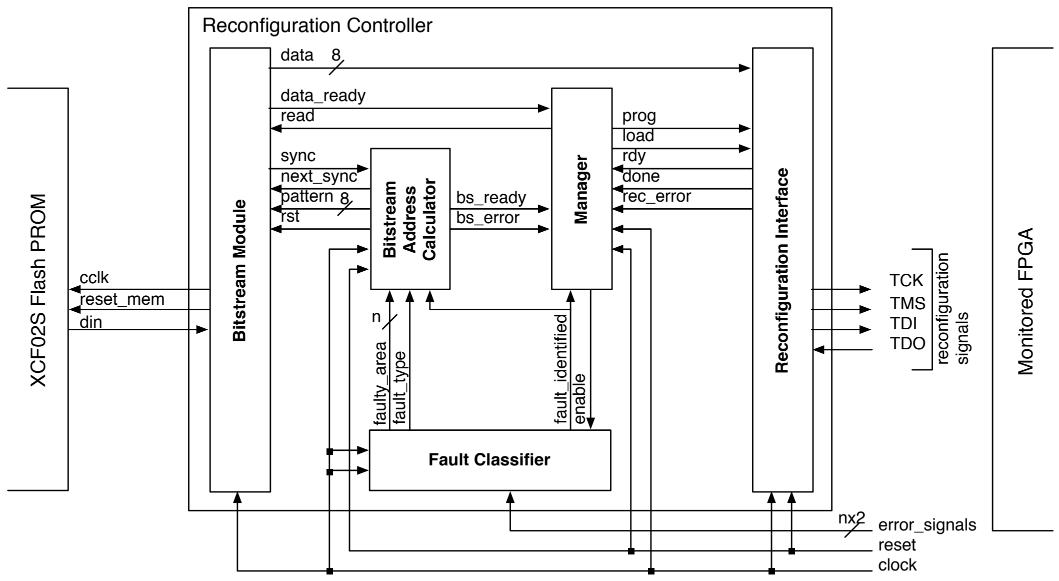

Fig. 7. Reconfiguration Controller block diagram.

The Bitstream Address Calculator is in charge of identifying the memory position of the configuration data for fault recovery. When the `fault_identified` signal is asserted, the module locates the suitable recovery bitstream based on `fault_type` and `faulty_area` information. As soon as the memory position has been identified, the `bs_ready` signal is asserted. When the number of non-recoverable faults exceeds the supported threshold, the recovery bitstream can not be located; in this case, the Bitstream Address Calculator signals an error by asserting `bs_error`.

The Bitstream Module is in charge of retrieving the recovery bitstream from memory. Based on the position information provided by the Bitstream Address Calculator and the reading request from the Manager, it retrieves the configuration data and sends them to the Reconfiguration Interface.

The Manager is the module that, upon error detection, moves the bitstream retrieved by the Bitstream Module to the Reconfiguration Interface. When the Fault Classifier raises the `fault_identified` signal, the Manager disables the classifier module through the `enable` signal, indicating that the error signals are not to be considered valid during the recovery phase. If the fault is non-recoverable, identified by the `fault_type` signal, the spare region devoted to relocation is checked by means of structural test before performing reconfiguration. The test phase is started by the `test_req` signal. When the `test_done` signal is asserted, if the test has been performed successfully, the Manager waits for the assertion of the `bs_ready` signal, specifying when the bitstream is ready to be retrieved. If the bitstream for recovering from the occurred fault can not be located, an error signal is asserted by the Bitstream Address Calculator. Otherwise, as soon as the configuration data can be read, the module asserts the `prog` signal to begin the reconfiguration process and starts the bitstream transfer from the Bitstream Module to the Reconfiguration Interface. It manages the data move between the two modules by using the `read` and `data_ready` signals of the Bitstream Module and the `load` and `rdy` signals of the Reconfiguration Interface. When the transfer is completed, it de-asserts the `prog` signal and waits for the assertion of the `done` signal. If an error occurs during the reconfiguration process, an error signal is asserted by the Reconfiguration Interface.

The Reconfiguration Interface performs the physical implementation of the recovery process, by providing access to the configuration interface (e.g., JTAG). The module is controlled by the Manager and receives the configuration data from the Bitstream Module. It handles the functions needed for programming. The recovery action by means of reconfiguration requires few seconds, and even few milliseconds in case of recoverable fault, being the typical configuration time around one microsecond per bit [Alfke 1998].

7.2. Controller hardening

The reliability of the Reconfiguration Controller must be guaranteed to block erroneous reconfigurations of the monitored FPGA and to include the module into an overall reliable system. Thus, it is necessary to detect faults affecting the controller and to guarantee the correctness of the reconfiguration process.

Particular attention has been devoted to the Fault Classifier, for which different hardened implementations have been analyzed, for the criticality of the classification task, as described in [Bolchini and Sandionigi 2011]. A self-checking implementation based on the application of error detection codes has been selected, leading to a competitive solution, characterized by acceptable costs and benefits in terms of reliability.

The hardening of the other modules composing the Reconfiguration Controller requires the introduction of fault detection and tolerance properties. Fault tolerance is guaranteed for the Reconfiguration Interface, because the correctness of the reconfiguration must be enforced. For the other components, i.e., Manager, Bitstream Address Calculator, and Bitstream Module, fault detection suffices since by detecting erroneous situations, it is possible to block erroneous reconfigurations. Thus, the selected hardening strategy applies TMR on the Reconfiguration Interface and duplication with comparison on the other components.

The costs of the hardened implementation are reported in Table III, for an implementation on a Xilinx Virtex4 xc4vlx100. Altogether, the controller requires a limited amount of the available resources, even in the hardened version, making the proposed solution a robust and affordable one.

Table III. Area occupation of the Reconfiguration Controller's hardened components.

Component	Slices	FFs
Fault Classifier	53	33
Bitstream Address Calculator	31	20
Bitstream Module	122	92
Manager	35	14
Reconfiguration Interface	912	728
Total	*1153*	*887*

8. METHODOLOGY EVALUATION

The reliability-aware design methodology presented in the paper has been evaluated by implementing a real case study. Section 8.1 describes the selected case study and Section 8.2 reports the experimental results. Finally, Section 8.3 compares the proposed approach against a solution achievable by adopting a methodology exploiting TMR and dynamic partial reconfiguration although devised for systems hosted on a single FPGA ([Bolchini et al. 2011a]), to see benefits and possible limitations.

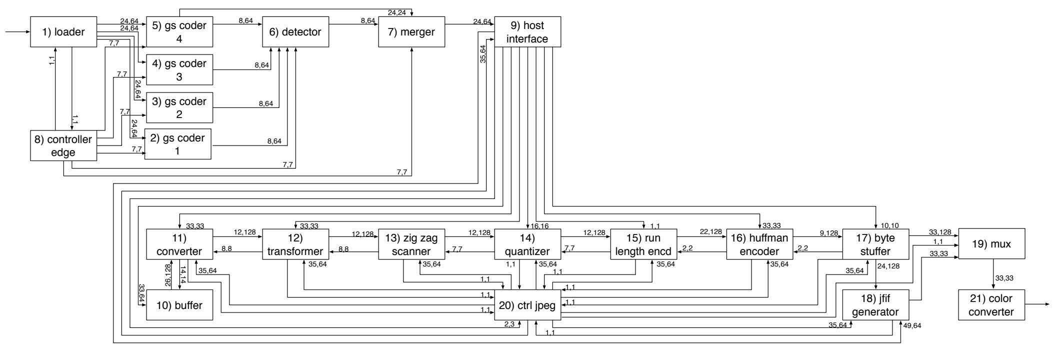

Fig. 8. Case study system.

8.1. Case study

The selected multi-FPGA platform is a commercial board, a Synopsys HAPS-34 [Synplicity]. It is composed of four Xilinx FPGAs xc4vlx100, interconnected to each other in a mesh topology through connections for communication and configuration.

The selected case study considers a circuit that performs image edge detection, to be used by an overall control system for people/objects recognition. It detects the image edges and converts the image to transfer it to different devices. In a first phase, the edges are detected and overlapped on the image. Then, the obtained raw bitmap image is encoded into a JPEG compliant coded bitstream, selected as standard format. Finally, as color conversion is necessary when transferring data between devices that use different color models, the image is converted to another color system. In Figure 8, the nominal circuit structure is shown, annotated with the requirements $< area\_wires_{a_1,a_2}$, $area\_comm_{a_1,a_2} >$ for the communication between modules; the throughput $area\_comm_{a_1,a_2}$ is identified by the number of bits processed at a time. The resource requirements of the circuit are reported in Table IV, by considering the implementation onto the selected device.

Table IV. Resource requirements of the case study circuit.

#	Component	#Slices	#BRAMs	#DSPs
1	loader	2341	12	-
2	gs coder 1	130	-	-
3	gs coder 2	135	-	-
4	gs coder 3	120	-	-
5	gs coder 4	105	-	-
6	detector	41	-	-
7	merger	17	-	-
8	controller edge	12	-	-
9	host interface	160	-	-
10	buffer	2212	12	-
11	converter	3512	-	-
12	transformer	2046	-	-
13	zig zag scanner	46	2	-
14	quantizer	76	3	1
15	run length encoder	266	2	-
16	huffman encoder	3560	-	-
17	byte stuffer	63	-	-
18	jfif generator	89	1	-
19	mux	19	-	-
20	controller jpeg	200	-	-
21	color converter	2435	-	-
Total		*17585*	*32*	*1*

8.2. Experimental results

The circuit components have been distributed among the devices available on the selected multi-FPGA platform. In the preliminary partitioning task, we aim at achieving a solution uniformly distributed among the available FPGAs, to have balanced ratio between used and spare fabric for applying the hardening techniques, thus preferring the distribution uniformity with respect to the other metrics. A tuning of the weights for the objective function has been performed and the following weights have been adopted:$w_{gap} = 0.7$, $w_{wires} = 0.1$, and $w_{comm} = 0.2$. The output of the partitioning is shown in Figure 9, together with the parameters characterizing the identified solution. The proposed approach achieves a uniform distribution of the circuit on the platform, also taking into account the external communication both in terms of wires and throughput. The partitioner's execution time for distributing the case study circuit among the four available devices is 33 s.

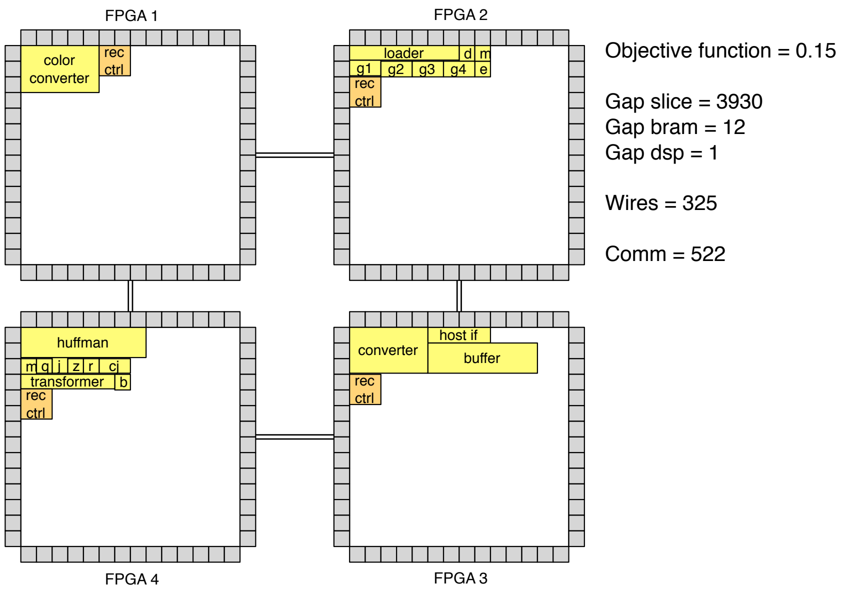

Fig. 9. System partitioning output.

Once the circuit has been partitioned among the available FPGAs, each sub-circuit is hardened by applying TMR to single components or groups of components, to define the IRAs. In particular, the design space exploration phase of the hardening task, determines the number of non-recoverable faults tolerated by the system on each FPGA ($\#faults$) and the maximum number of IRAs (max_areas) per FPGA device, considering a 256 MB memory to store bitstreams. The computed values for the given case study are $\#faults$ = 2 and max_areas = 7, respectively.

Table V reports, for each FPGA, how components are grouped together for an efficient application of the TMR, showing also the independently recoverable areas of each group, and the resource requirements of each IRA. Table VI specifies the hardware overhead required by each group of components. A tuning of the weights for the objective function has been performed and the following weights have been adopted for hardening the sub-circuits: w_{res} = 0.6, w_{areas} = 0.3, and w_{wires} = 0.1. Thus, the distribution uniformity is achieved, also taking into account the maximization of the number of IRAs, as required to apply the fault classification algorithm. Moreover, the minimization of the external wires is considered, to avoid solutions with groups where components are not connected to each other. The average execution time for exploring the solution space in order to harden each sub-circuit is 3 s.

Table V. Definition of the IRAs for each FPGA.

FPGA 1					
Group	Components	Area	#Slices	#BRAMs	#DSPs
1	color converter	1	2435	-	-
		2	2435	-	-
		3	2567	-	-
FPGA 2					
Group	Components	Area	#Slices	#BRAMs	#DSPs
2	loader, gs coder 1, 2, 3, 4,	1	2901	12	-
	detector, merger	2	2901	12	
	controller edge	3	2997	12	
FPGA 3					
Group	Components	Area	#Slices	#BRAMs	#DSPs
3	host interface, buffer	1	2372	12	-
		2	2372	12	-
		3	3184	12	-
4	converter	1	3512	-	-
		2	3512	-	-
		3	3620	-	-
FPGA 4					
Group	Components	Area	#Slices	#BRAMs	#DSPs
5	transformer, zig zag scanner,	1	2805	8	1
	quantizer, run length encoder	2	2805	8	1
	byte stuffer, jfif generator,	3	3345	8	1
	mux, controller jpeg				
6	huffman	1	3560	-	-
		2	3560	-	-
		3	3608	-	-

Table VI. Overhead cost in terms of slices for each group of components.

Group	Nominal	Hardened	Overhead
1	2435	7437	+205%
2	2901	8799	+203%
3	2372	7928	+234%
4	3512	10644	+203%
5	2805	8955	+219%
6	3560	10728	+201%

By taking into account the defined hardened circuit, the number of tolerated non-recoverable faults is re-evaluated by the refinement task. For the selected case study, the number of tolerated non-recoverable faults is $\#faults = 2$, confirming the preliminary evaluation. In different case studies, the hardening process actually requires a reduced amount of resources with respect to those estimated in the preliminary phase, thus allowing to achieve higher value of $\#faults$, corresponding to an actual longer system lifetime.

The refinement task also re-distributes the IRAs among the FPGAs, to possibly obtain a better partitioning. Indeed, the areas belonging to the same group are constrained to be hosted on the same FPGA. Again, in the present case study, the same solution obtained by the preliminary partitioner is achieved.

The floorplanning activity positions each hardened sub-circuit within the assigned FPGA. Figure 10 shows the floorplan of each sub-circuit. This task guarantees the feasibility of the identified solution and produces the positioning constraints for implementing the system on the multi-FPGA platform. Finally, the defined hardened circuit is integrated with the Reconfiguration Controllers (one for each FPGA), creating the envisioned reliable system.

The experiments have been executed on an Intel Nehalem dual socket quad-core system with 32 GB of RAM. The design flow has required almost one minute for performing the various explo-

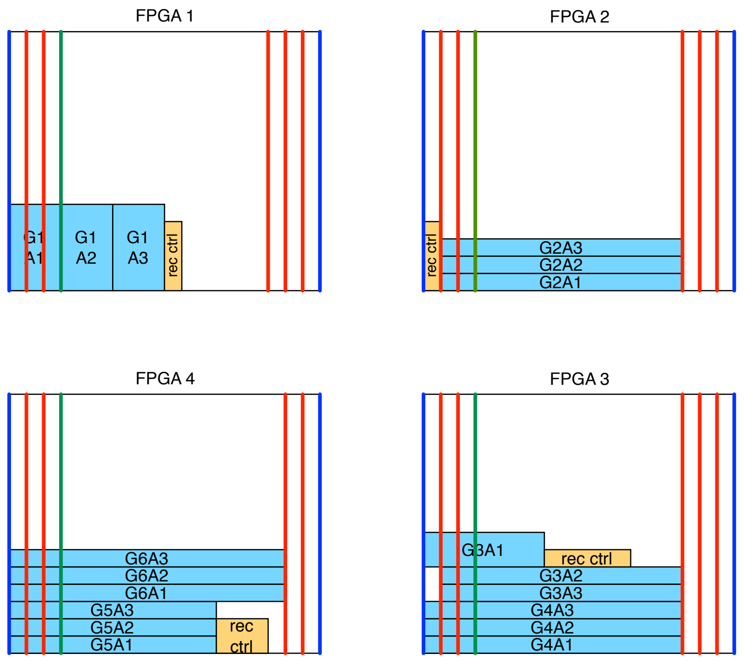

Fig. 10. Floorplan for each FPGA.

rations, namely partitioning, independently recoverable areas definition, and floorplanning. It produces the configurations for implementing the system on the FPGAs and the recovery bitstreams.

Finally, by referring to the data reported in [Bolchini et al. 2011c], we can provide a rough estimation of the lifetime expected for the considered system. Without applying hardening techniques, the system could survive only few months, since the MTBF of recoverable faults is estimated as few months (even only a couple of months) also at Low-Earth Orbit. When considering non-recoverable faults, the minimum identified MTBF is about 400 days, due to TDDB. By applying our approach, the obtained hardened system can cope with recoverable faults and can tolerate up to 2 non-recoverable faults, hence the expected lifetime is considerably improved.

8.3. Comparison with related work

To the best of the authors' knowledge, there are no other approaches in literature for designing hardened systems on multi-FPGA platforms, while often an ad-hoc solution is adopted. One straightforward solution consists in designing three replicas of the same FPGAs and voting the final outputs, as proposed in [Smith and de la Torre 2006], however the solution is not flexible with respect to the size of the system, and more in general with the hardening strategies that can be adopted for the IRAs. Indeed, it could be possible to tackle the problem by considering a virtual FPGA having an amount of resources equal to the total resources of the n FPGAs, and later partition the resulting system on the different boards. Here we pursue this kind of solution, to evaluate the benefits of a more refined methodology, taking into account the real platform. In particular, we exploited the framework presented in [Bolchini et al. 2011a] by mimicking a bigger device (namely one composed of the resources offered by four xc4vlx100 FPGAs). Pin constraints and resource distribution have been adjusted to resemble the real final platform, so that the framework could provide a usable solution, requiring the designer only to manually position and distribute the components onto the FPGAs. Nevertheless, the framework identified 31 solutions with respect to the bigger fabric, none of which actually feasible on the multi-FPGA platform. Figure 11 reports one of the identified solutions.

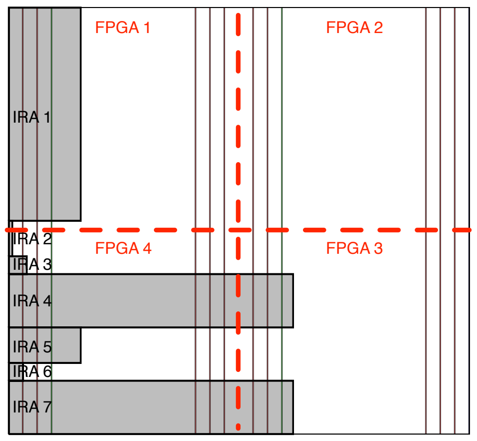

Fig. 11. Solution obtained with the approach in [Bolchini et al. 2011a].

As expected, the independently recoverable areas are positioned also on the intersection between different devices and only onto a portion of the available fabric, thus exploiting only a couple of the devices and without achieving a balance between used and spare regions. The application of the partitioning task (the one used as a refinement) at such an unbalanced solution would though lead to poor solutions, highlighting the need for a specific multi-FPGA flow, such as the one we propose.

9. CONCLUSIONS

This paper has proposed a complete methodology for the design of self-healing embedded systems on multi-FPGA platforms. The aim of the work is the exploitation of commercial SRAM-based FPGAs for mission-critical applications. The proposed methodology leads the designer in the realization of a multi-FPGA system able to detect and cope with the occurrence of faults globally and autonomously, thus extending its lifetime and availability. A circuit hardening approach based on a hybrid strategy has been adopted; the circuit is hardened by means of traditional reliability techniques, e.g. exploiting space redundancy, and a reconfiguration of the FPGAs is used to mitigate fault effects. We have identified two categories of faults, based on the possibility to recover from them by reconfiguration: Recoverable faults, that can be mitigated by reconfiguring the system (and possibly only the faulty sub-system portion) with the same configuration used before fault occurrence, and non-recoverable faults, that are caused by a destructive effect and lead to the necessity of relocating the functionality to a non-faulty region of the device. In the envisioned reliable system, each FPGA hosts i) a hardened portion of the entire circuit, organized in independently recoverable areas that detect, mask/tolerate, and signal the occurrence of a fault, and ii) a Reconfiguration Controller, in charge of monitoring the error signals and, when needed, performing the suitable reconfiguration of the faulty part based on the type of fault. The obtained autonomous fault tolerant system can continue working even if faults occur, thus increasing its lifetime. It is worth noting that the solution we pursue is not a generic one, rather it is identified by means of a design space exploration, to evaluate different trade-offs, meeting the designer's interests and constraints.

REFERENCES

ALFKE, P. 1998. Xilinx FPGAs: A technical overview for the first-time user. Tech. Rep. XAPP097, Xilinx Inc.

ARMIN, A., MAHNAZ, S. Y., AND ZAINALABEDIN, N. 2006. An optimum ORA BIST for multiple fault FPGA Look-Up Table testing. In *Proc. Asian Test Symposium*. 293–298.

ATF280E 2007. Rad Hard Reprogrammable FPGA. A. Corporation.

BOLCHINI, C., FOSSATI, L., CODINACHS, D. M., MIELE, A., AND SANDIONIGI, C. 2010. A reliable reconfiguration controller for fault-tolerant embedded systems on multi-FPGA platforms. In *Proc. Symp. Defect and Fault Tolerance in VLSI Systems*. 191–199.

BOLCHINI, C. AND MIELE, A. 2008. Design space exploration for the design of reliable SRAM-based FPGA systems. In *Proc. Symp. Defect and Fault Tolerance in VLSI Systems*. 332–340.

BOLCHINI, C., MIELE, A., AND SANDIONIGI, C. 2011a. A novel design methodology for implementing reliability-aware systems on SRAM-based FPGAs. *IEEE Trans. Computers 60,* 12, 1744–1758.

BOLCHINI, C., MIELE, A., AND SANDIONIGI, C. 2011b. Automated resource-aware floorplanning of reconfigurable areas in partially-reconfigurable FPGA system. In *Proc. Int. Conf. Field Programmable Logic and Applications*. 532–538.

BOLCHINI, C. AND SANDIONIGI, C. 2010. Fault classification for SRAM-based FPGAs in the space environment for fault mitigation. *IEEE Embedded Systems Letters 2*, 107–110.

BOLCHINI, C. AND SANDIONIGI, C. 2011. A reliability-aware partitioner for multi-FPGA platforms. In *Proc. Symp. Defect and Fault Tolerance in VLSI Systems*. 34–40.

BOLCHINI, C., SANDIONIGI, C., FOSSATI, L., AND CODINACHS, D. M. 2011c. A reliable fault classifier for dependable systems on SRAM-based FPGAs. In *On-Line Testing Symposium (IOLTS), 2011 IEEE 17th International*. 92–97.

CARMICHAEL, C., CAFFREY, M., AND SALAZARI, A. 2000. Correcting Single-Event Upsets through Virtex partial configuration. Tech. Rep. XAPP216, Xilinx Inc.

CARMICHAEL, C. AND TSEN, C. W. 2009. Correcting Single-Event Upsets in Virtex-4 FPGA configuration memory. Tech. Rep. XAPP1088, Xilinx Inc.

ECSS-E-ST-10-12C 2008. Methods for the calculation of radiation received and its effects, and a policy for design margin. European Cooperation for Space Standardization.

EMMERT, J., STROUD, C., SKAGGS, B., AND ABRAMOVICI, M. 2000. Dynamic fault tolerance in FPGAs via partial reconfiguration. In *Field-Programmable Custom Computing Machines, 2000 IEEE Symposium on*. 165–174.

FOSSATI, L. AND ILSTAD, J. 2011. The future of embedded systems at ESA: Towards adaptability and reconfigurability. In *Proc. NASA/ESA Adaptive Hardware and Systems*. 113–120.

HAUCK, S. AND DEHON, A. 2007. *Reconfigurable Computing: The Theory and Practice of FPGA-Based Computation*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Lattice Semiconductor Corporation 2010. FPGA Devices. Available: http://www.latticesemi.com/products/fpga/index.cfm.

MAY, T. AND WOODS, M. 1979. Alpha-particle-induced soft errors in dynamic memories. *IEEE Trans. Electron Devices ED-26,* 1, 2–9.

MITRA, S., HUANG, W.-J., N. R. SAXENA, S.-Y. Y., AND MCCLUSKEY, E. 2004. Reconfigurable architecture for autonomous self-repair. *IEEE Design & Test of Comp. 21,* 3, 228–240.

MONTMINY, D. P., BALDWIN, R. O., WILLIAMS, P. D., AND MULLINS, B. E. 2007. Using relocatable bitstreams for fault tolerance. In *Proc. NASA/ESA Conf. Adaptive Hardware and Systems*. 701–708.

MORGAN, K. S. 2006. SEU-induced persistent error propagation in FPGAs. Ph.D. thesis, Brigham Young University.

PETRICK, D., POWELL, W., JR., J. W. H., AND LABELI, K. A. 2004. Virtex-II Pro SEE test methods and results. In *Proc. Military and Aerospace Applications of Programmable Devices and Technologies Conf.*

POIVEY, C., BERG, M., STANSBERRY, S., FRIENDLICH, M., KIM, H., PETRICK, D., AND LABEI, K. 2007. Heavy ion SEE test of Virtex4 FPGA XC4VFX60 from Xilinx. Tech. Rep. T021607¯XC4VFX60.

RENOVELL, M. 2002. A structural test methodology for SRAM-based FPGAs. In *Proc. Symp. Integrated Circuits and Systems Design*.

SAMUDRALA, P. K., RAMOS, J., AND KATKOORI, S. 2004. Selective Triple Modular Redundancy (STMR) based Single-Event Upset (SEU) tolerant synthesis for FPGAs. *IEEE Trans. Nuclear Science 51,* 5, 2957–2969.

SMITH, G. L. AND DE LA TORRE, L. 2006. Techniques to enable FPGA based reconfigurable fault tolerant space computing. In *Proc. Aerospace Conference*. 1–11.

SRINIVASAN, S., KRISHNAN, R., MANGALAGIRI, P., XIE, Y., NARAYANAN, V., IRWIN, M. J., AND SARPATWARII, K. 2008. Toward increasing FPGA lifetime. *IEEE Embedded Systems Letters 5,* 2, 115–127.

SRINIVASAN, S., MANGALAGIRI, P., XIE, Y., VIJAYKRISHNAN, N., AND SARPATWARI, K. 2006. FLAW: FPGA Lifetime AWareness. In *Proc. Design Automation Conf.* 630–635.

Synplicity. HAPS-34. http://www.synopsys.com/home.aspx.

Xilinx Inc. 2006. TMRTool. http://www.xilinx.com/esp/mil¯aero/collateral/tmrtool¯sellsheet¯wr.pdf.