2010 25th International Symposium on Defect and Fault Tolerance in VLSI Systems

A Reliable Reconfiguration Controller for Fault-Tolerant Embedded Systems on Multi-FPGA platforms

C. Bolchini*, L. Fossati†, D. Merodio Codinachs†, A. Miele* and C. Sandionigi*
*Dip. di Elettronica e Informazione, Politecnico di Milano
P.zza L. da Vinci, 32 - 20133 Milano, Italy
Email: {bolchini, miele, sandionigi}@elet.polimi.it
†European Space Agency
Keplerlaan, 1 - 2200 AG Noordwijk, The Netherlands
Email: {Luca.Fossati, David.Merodio.Codinachs}@esa.int

Abstract

This paper proposes the design of a controller managing the fault tolerance of multi-FPGA platforms, contributing to the creation of a reliable system featuring high flexibility and resource availability. A fault management strategy that exploits the devices' reconfiguration capabilities is proposed; the Reconfiguration Controller, focus of this paper, is the main component in charge of implementing such strategy. The innovative points raised by this work are 1) the identification of a distributed control architecture, allowing the avoidance of single points of failure, 2) the management of both recoverable and non-recoverable faults, and 3) the definition of an overall reliable multi-FPGA system.

Keywords

Hard/Soft Errors, Fault Tolerance, Multi-FPGA platforms, Reconfiguration

I. Introduction

During the last decade, Field Programmable Gate Arrays (FPGAs) have been adopted as a target technology for both the fast prototyping and final production of embedded systems, due to their low cost, ease of use and, in particular, flexibility, related to the opportunity of re-programming (or *reconfiguring*) the device in a few clock cycles and on-line, while the device is operating. Among the many different FPGA families, SRAM-based ones are the most used platforms, as they allow easy and efficient exploitation of the device's reconfiguration property. Moreover, such FPGA family is characterized by high density and high operational frequency, with a limited cost with respect to other solutions [1]. Given their peculiarities, SRAM-based FPGA platforms' use is increasing not only in the standard commercial market, but also in mission- or safety-critical applications, where the occurrence of failures could cause serious hazards to the users and the environment; for instance, the space environment, considered as a possible application scenario for this work, is particularly critical as maintenance is not affordable, thus fault mitigation and recovery techniques are paramount to guaranteeing that the system will continue working correctly for the entire duration of the mission. Even if SRAM-based FPGAs are more susceptible to faults with respect to traditional solutions (e.g., ASICs) [2], they have higher capabilities of recovering from them, thanks to their reconfigurability properties; when a fault is detected, the affected functionality is re-created on the FPGA device, thus restoring the device overall processing capabilities. Another aspect gaining importance is the limited amount of functionalities they can accommodate; indeed, as the size and complexity of the systems being designed increase, a single FPGA may not suffice in terms of available resources, and multi-FPGA solutions start being taken into account and investigated (e.g., [3]–[6]).

In this paper, we focus on the reliability aspects related to the implementation of systems onto multi-FPGA platforms, where fault mitigation still needs to be investigated. In particular, while fault detection and masking techniques have been widely addressed in the case of systems based on a single FPGA (e.g., [7]–[9]), the problem has not been yet extended to multi-FPGA platforms. Together with giving an overview of the proposed reliable multi-FPGA system, in this paper we introduce the design of a controller in charge of implementing the fault mitigation strategy, an aspect seldom discussed in literature. While there are some fault detection and mitigation strategies and they could be adapted to the multi-FPGA scenario, we claim that an ad-hoc solution for this scenario would consistently improve the system's performance and its reliability. In particular, as discussed in Section II, those studies proposing a reconfiguration controller architecture (e.g., [5], [10]–[13]) focus the attention on the efficiency of the partial, self-reconfiguration aspects, and do not address reliability-related issues.

This work is partially supported by ESA/ESTEC Contract #22079/08/NL/JK and by Italian MIUR-PRIN project #2008K4P7X9.

1550-5774/10 $26.00 © 2010 IEEE
DOI 10.1109/DFT.2010.30

Therefore, we here introduce a reliability-aware reconfiguration controller, aimed at managing the reconfiguration process of the architecture parts to mitigate fault effects in multi-FPGA platforms.

The paper is organized as follows: related work on controller design is presented in the next section, while Section III gives an overview of the approach used to implement a reliable embedded system on a multi-FPGA platform, introducing all elements characterizing the working scenario. Section IV presents the Reconfiguration Controller design, evaluated in Section V discussing its peculiarities and strong points. Finally, Section VI draws the conclusions and identifies future work.

II. Related work

To the best of our knowledge, the system closer to the one being described in this paper has been proposed by Honeywell in [5]. It consists of a controller for a system composed of three FPGAs, each one hosting the same configuration, where the controller, implemented on an external radiation-hardened ASIC, acts as a Triple Modular Redundancy (TMR) voter. The voter sub-system, upon error detection, reconfigures the corrupted FPGA and re-synchronizes it with the two remaining FPGAs. The multiple FPGAs are here only used to host replicas of the main system, exploiting the classical TMR technique for system hardening, which is a different scenario with respect to the one we envision. By applying TMR at finer granularity level, we better exploit the devices potential and we provide a scalable solution, independent of the number of FPGAs used. In addition, Honeywell's controller needs to be implemented on particular ASIC technology, eliminating most of the advantages of having FPGAs in the system, namely flexibility and relatively low cost. Finally, the solution can cope with recoverable faults but does not address non-recoverable ones. Central in Honeywell's paper, as in our work, is the controller module, necessary for an appropriate management of the overall system functionality and responsible for detecting faults and correcting them by reconfiguring the FPGAs' functionalities.

Literature reports a few controller designs, however most of them focusing on dynamic partial self-reconfiguration for single FPGA platforms, without taking reliability issues into account [10]–[13]. More precisely, the authors of [10] propose a solution composed of a microprocessor, that manages the reconfiguration process, and of a controller for the Internal Configuration Access Port (ICAP) interface [14], [15], that enables on-chip dynamic partial reconfiguration working close to the maximum achievable speed. The microprocessor sends the reconfiguration request to the ICAP controller through a Device Control Register bus, and the ICAP controller accesses a memory to fetch the bitstream for reconfiguration by performing Direct Memory Access (DMA) via a Processor Local Bus (PLB). Besides introducing such architecture for dynamic partial reconfiguration, the authors introduce a method to calculate the expected reconfiguration throughput and latency. The use of DMA and burst transfers allow the presented reconfiguration controller to work at a speed that is close to the theoretical maximum achievable throughput. A preliminary version of this reconfiguration controller exclusively designed for the Xilinx Virtex-II Pro [14] FPGA is presented in [11]. In both versions, particular relevance is given to the ICAP controller, constituting an essential aspect in the achievement of the obtained performance; in fact, compared to an alternative realization using a state-of-the-art solution, namely the OPB HWICAP IP-Core [16], a speed-up by a factor of 20 has been achieved by the older version, and a speed-up by a factor of 58 has been obtained by the new one.

A customized version of this reconfiguration controller is shown in [12], where the authors describe a framework used to create the most suitable controller according to the reconfiguration scenario where it will be used. A set of metrics has been defined to describe the reconfiguration scenario and to set the following parameters of the reconfiguration controller: i) the bus interface of the ICAP controller, by allowing to choose between the PLB and the On-chip Peripheral Bus, such that the problem of having the ICAP controller on a bus and the memory on another one no longer exists; ii) the implementation type, in terms of slices or BRAMs, of the memory used to store the bitstream inside the ICAP controller, in order to set the resources requirement; finally, iii) the size of the internal memory, by allowing to find a trade-off between resources requirement and reconfiguration throughput. Thus, the attention has been oriented towards the speed-up of the reconfiguration process, as in the previous two reconfiguration controllers, and its simplification.

Another type of controller, implemented by a stand alone IP-Core, is presented in [13]; this custom soft core, called Parallel Configuration Access Port (PCAP), performs partial dynamic self reconfiguration through the SelectMAP port, by using a partial bitstream stored in a BlockRAM memory within the FPGA itself. The proposed approach stores partial bitstreams on on-chip memory and reads them from there under the control of PCAP. In this case, the reconfiguration process is accomplished without ICAP, present only in some FPGA families, and without a microprocessor managing the process.

While the analyzed solutions do not take into account the system fault tolerance and only concentrate on a single FPGA, they are the first proposals of efficient reconfiguration controllers and, as such, they were carefully taken into account while defining the presented solution, possibly adopting the advantageous solutions they propose. However, our work deals with the problem of multi-FPGA platforms' reliability, therefore the attention is on the effects of faults and their mitigation rather than on the overall performance. Hence, we propose a fault management strategy that exploits the reconfiguration capability

of the devices to cope with the occurrence of both recoverable and non-recoverable faults. The component implementing the strategy, namely the *Reconfiguration Controller*, has been designed, identifying its functionality and features. A preliminary software implementation has been developed to validate the designed controller with respect to its requirements, and a hardware prototype is being developed. Innovative points of the presented solution are: i) distributed, low cost Reconfiguration Controller, eliminating the need to deploy it on ad-hoc device, ii) management of both recoverable and non-recoverable faults, and iii) scalable use of multiple FPGAs to build a reconfigurable reliable system.

III. The Reliable Multi-FPGA Scenario

This section details all the fundamental aspects of the proposed approach for implementing fault tolerant systems on multi-FPGA platforms, by introducing the system architecture, the adopted fault model, and the selected fault management strategy. These elements constitute the working scenario for the proposed Reconfiguration Controller.

A. System Architecture

The devised platform is composed of multiple SRAM-based FPGAs connected to each other in a *mesh topology*, i.e., each FPGA is connected to more than one FPGA and they can all connect to each other via multiple hops.

In the general situation, the complete *nominal*, not hardened, system is distributed onto the platform; partitioning of the application system among different FPGAs is here not addressed, since the proposed approach does not depend on it. When fault tolerance properties are required, a system hardening approach based on a hybrid strategy is adopted; the system is hardened by means of techniques exploiting space redundancy, and a reconfiguration of the devices is used to mitigate fault effects, as proposed, for example, in [8]. In this perspective, each FPGA hosts: i) a hardened portion of the entire system, organized in *Independently Recoverable Areas*, as detailed later on, to detect, mask/tolerate and signal the occurrence of a fault, and ii) a Reconfiguration Controller, in charge of monitoring the error signals to trigger, when needed, the reconfiguration of the faulty sub-system. An overview of the system architecture is shown in Figure 1.

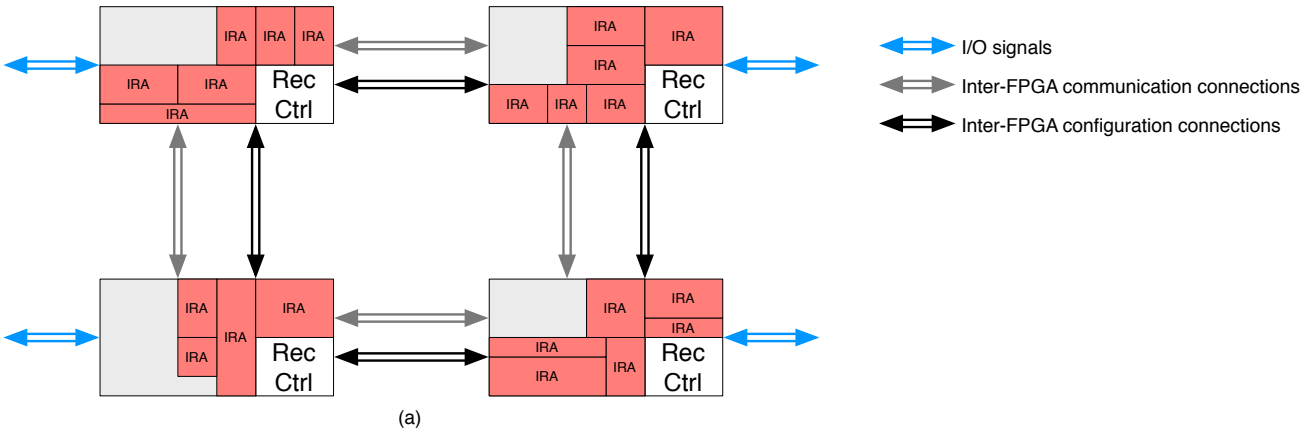

Figure 1. Proposed system architecture.

Rather than making each FPGA an independent fault-tolerant sub-system, able to locally detect and recover from faults, we have envisioned a distributed 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 single point of failure.

B. Adopted fault model

Two types of faults are here taken into account: recoverable and non-recoverable. The former are caused, for instance, by cosmic rays in space environment, and are usually taken into account considering their probability of occurrence [17]. The latter are caused by device aging [18], and in the field of FPGAs are just being recently addressed, also due to their rareness. Appropriate actions can be taken for faults in the *first category*, restoring the device to its nominal operation. In the context of FPGAs, such faults can be represented, for example, by a bit flip in the SRAM configuration memory, and they can be corrected by reconfiguring the device [9]. Within recoverable faults, we mainly focus on Single Event Effects, and specifically on Single Event Upsets (SEUs); when such faults affect the configuration memory, they can cause a modification in the system functionality, with a persistent effect until a re-writing of the correct configuration is performed.

Faults belonging to the *second category* permanently compromise part or all of the device; non-recoverable faults forbid further use of the corrupted portion of the device and the logic hosted must be *moved* in a different location.

In our scenario, the following assumptions are taken into account: i) faults occur *one at a time* within the whole system, and ii) the time between the occurrence of two subsequent faults is long enough to allow detection and recovery from the first fault before the second occurs. Note also that, while any number of recoverable faults can take place during the system's lifetime, only one non-recoverable fault per FPGA is tolerated.

Based on the described fault model and the related assumptions, the following fault management strategy is proposed.

C. Hardening strategy

The hardening of the system is achieved by a) applying TMR to the system components, such that their output is voted and fed to subsequent modules, and b) placing each of such hardened components in a portion of the FPGA, here called *Independently Recoverable Areas* (IRAs), that can be reconfigured independently by the remaining part of the device when a fault occurs. The granularity at which the TMR is applied can be decided by the designer, going from single flip-flops to the whole system; in general we found appropriate to divide the application into 10-15 macro-blocks, which are separately hardened (see [8] for more details). The TMR voter not only masks the effect of the fault by performing a majority vote, but it also outputs an error signal detecting a mismatch in the voted data or a fault in the voter itself. To increase the controller's robustness, such error signal is encoded with the Two-Rail Code (TRC) [19], using two output signals to denote either a fault-free situation or the occurrence of a fault in one of the replicas or in the voter. As a result, each IRA is a fault masking, Self-Checking sub-system, that can be recovered, when a recoverable fault occurs, by partially reconfiguring the FPGA. Should the fault be considered as non-recoverable, the IRA becomes tagged as "unusable" and the hosted sub-system will be re-located to one of the spare IRAs, reserved on purpose on each FPGA, by totally reconfiguring the FPGA; in this case, the whole FPGA functionality needs to be rewritten as not only the position of the functionality itself has changed, but also the routing resources could be different. This hardening strategy can be automatically carried out by adopting a methodology similar to the one presented in [8].

Fundamental element of the hardening strategy is the Reconfiguration Controller, in charge of monitoring the error signals coming from the IRAs of the neighbor FPGA; the controller's main task consists of triggering, when needed, the reconfiguration process, hence recovering from faults. Furthermore, the Reconfiguration Controller periodically accesses the neighbor FPGA to perform the partial readback of the configuration memory corresponding to the controller, to detect, and eventually repair, faults. The Reconfiguration Controller itself can be seen as an IRA that, in case of fault, is recovered or relocated based on the type of occurred fault.

An overview of the described system is reported in Figure 2, where the global hardened multi-FPGA architecture is depicted (Figure 2(a)) together with the detail of the reconfiguration approach (Figure 2(b)). In case of a fault of an IRA on $FPGA_{i+1}$, the Reconfiguration Controller hosted on $FPGA_i$ detects the anomaly, as shown in Figure 2 (a). The fault is classified as recoverable or non-recoverable and, based on the type of fault, the Reconfiguration Controller on $FPGA_i$ performs the reconfiguration using the correct bitstream, as shown in Figure 2 (b): if the fault is recoverable, the functionality is restored by performing partial reconfiguration, otherwise a relocation of the functionality is performed through total reconfiguration, moving the functionality on a non-faulty device region.

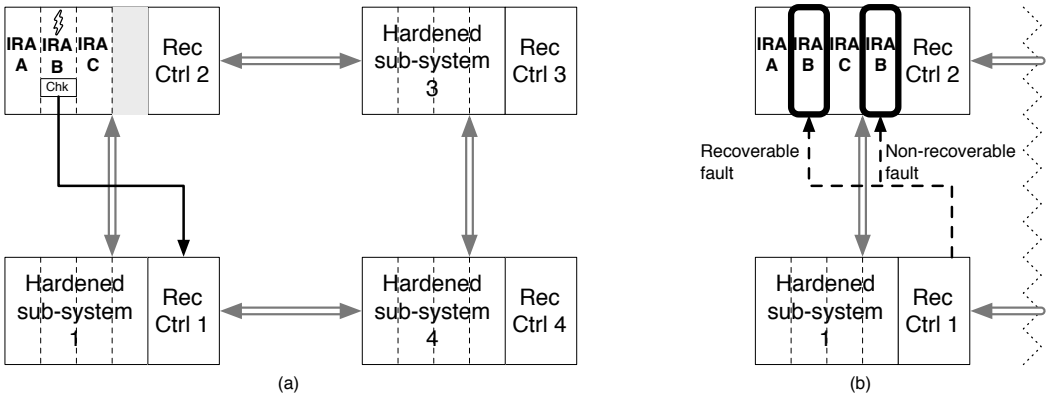

Figure 2. Target scenario: (a) hardened multi-FPGA architecture and (b) proposed fault management strategy.

To manage also faults occurrence on Reconfiguration Controller, in addition to monitoring and managing the reconfigurable functionalities of $FPGA_{i+1}$, the RC of $FPGA_i$ also checks the RC of $FPGA_{i+1}$ by performing a *partial readback* [20], that retrieves the RC configuration data from the internal configuration memory on $FPGA_{i+1}$, and compares it with a golden copy stored in a rad-hard memory.

The Reconfiguration Controller is the innovative proposal this paper focuses on, exploiting the fault management strategy, as detailed in the next section.

IV. The Reconfiguration Controller design

The Reconfiguration Controller is the main component responsible for the active fault mitigation process. Its main tasks are: i) continuously monitoring the neighbor FPGA error signals from the IRAs, ii) periodically monitoring the controller hosted on the neighbor FPGA, and iii) upon error detection in one of the areas, IRAs or controller, trigger a reconfiguration, based on the classification of the fault nature (recoverable vs. non-recoverable). The flow diagram shown in Figure 3 reports the behavior of the Reconfiguration Controller.

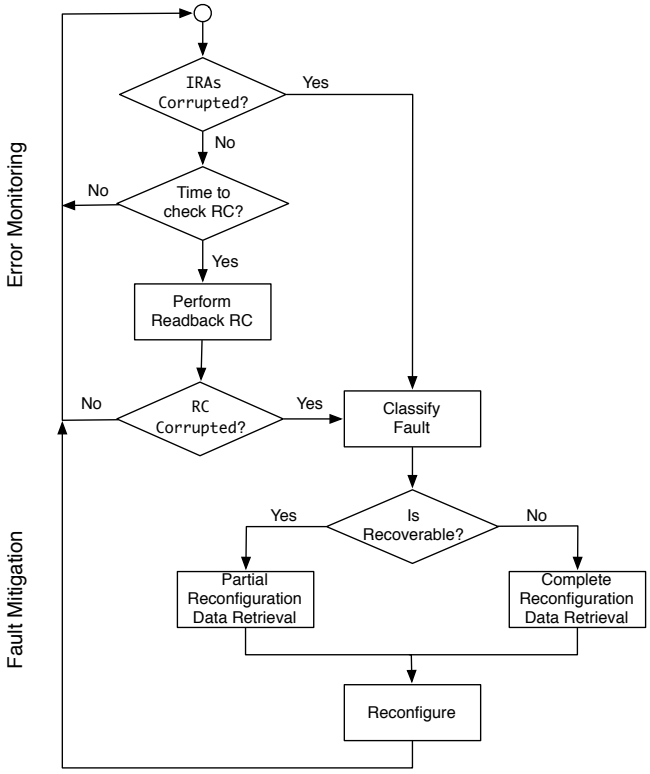

Figure 3. Reconfiguration Controller flow diagram

In order to carry out such tasks, the following modules have been identified as fundamental "building blocks": a *Reconfiguration Interface*, in charge of performing the physical implementation of the reconfiguration process, providing access to the FPGA configuration memory interface, and a *Manager*, to control the Reconfiguration Interface based on error detection and classification policies, determining what bitstream shall be used to reconfigure the platform and mitigate (recover or relocate) the effect of the fault. The Manager is, in turn, composed of the following elements: i) a *Readback Module*, for the neighbor controller verification, ii) a *Fault Classifier*, to determine whether the detected fault can be considered recoverable or non-recoverable, and iii) a *Bitstream Module*, that retrieves the data and information to perform the necessary reconfiguration. Note how the Reconfiguration Interface is a bottleneck for the overall system reliability as a fault in it could trigger undesired reconfiguration, possibly corrupting the functionalities of the neighbor FPGA; hence, even if there is no explicit self-analysis of the absence of faults in the Reconfiguration Controller, the controller itself is designed to avoid such undesired reconfigurations: when the reconfiguration request is asserted, the error signals are checked to detect if an error has actually been raised, otherwise the reconfiguration is blocked.

An overview of the resulting Reconfiguration Controller structure is presented in Figure 4, and the detailed presentation of the Manager module is proposed in the next subsections.

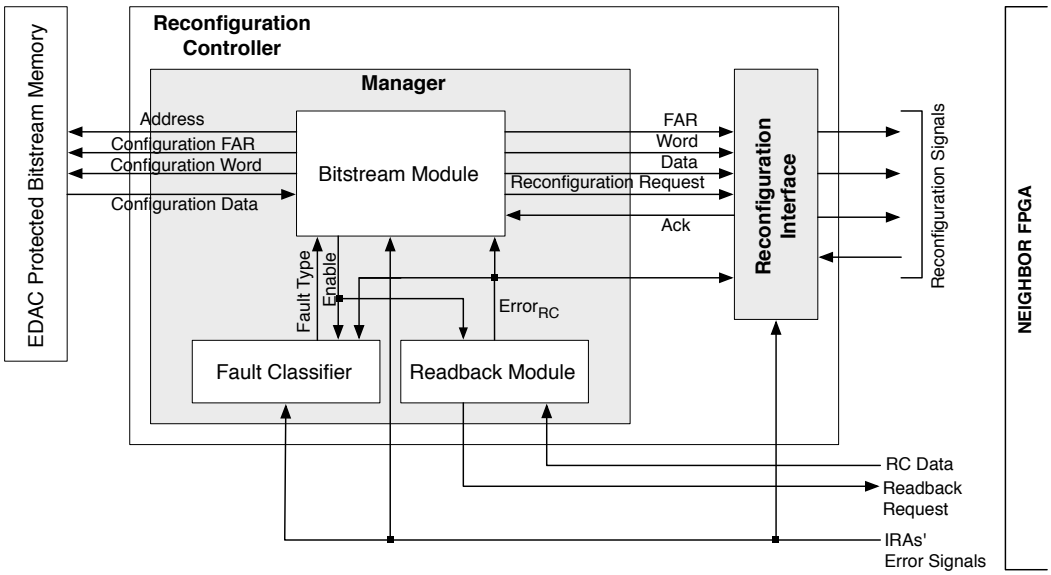

Figure 4. Reconfiguration Controller block diagram

A. Readback Module

While the hardened system organized in IRAs, as described in Section III, is designed to be self-checking, i.e., it incorporates logic to detect the presence of a fault causing an error in its functionalities, this is not true for the Reconfiguration Controller itself. As explained, the Reconfiguration Controller is designed to behave correctly even if a fault occurs in the portion of the FPGA hosting it; however it is the neighbor Reconfiguration Controller's task, through the Readback Module, to determine the presence of such a fault.

We adopt the following strategy: the presence of a fault in the Reconfiguration Controller of $FPGA_{i+1}$ is determined by the Reconfiguration Controller of $FPGA_i$, which periodically reads the configuration data and compares it with a golden model stored in a protected memory. Such operation is performed every T_{rb} time instants, estimated on the Mean Time Between Failures (MTBF) of the adopted fault model in the envisioned application scenario, mimicking the methodologies presented in [9]. Furthermore, an *Enable* signal, coming from the Bitstream Module, is provided in input, to block the readback activity while the neighbor FPGA is being reconfigured to correct a previously detected error.

The output of the comparison, $\texttt{Error}_{\texttt{RC}_{i+1}}$, is encoded with the classical TRC, to provide an adequate level of protection within the Reconfiguration Controller module, hence preventing internal faults from triggering unnecessary/incorrect reconfigurations of the monitored neighbor controller $\texttt{RC}_{\texttt{i+1}}$. Such output signals are sent to the Fault Classifier, in charge of classifying the fault as transient or permanent.

B. Fault Classifier

The Fault Classifier module receives the error signals from the Readback Module ($\texttt{Error}_{\texttt{RC}_\texttt{i}}$) and from the monitored n IRAs ($\texttt{Error}_{\texttt{IRA}_1}, \cdots, \texttt{Error}_{\texttt{IRA}_\texttt{n}}$) of $FPGA_{i+1}$, and, when an error is detected, it classifies the fault as recoverable or non-recoverable. In detail, the input of the module is a $(n+1) \times 2$ bits signals, since each error signal is encoded with the TRC. Furthermore, as in the Readback Module, an `Enable` signal, coming from the Bitstream Module, is provided to specify when the error signals are to be considered valid, or not. Fault classification must be disabled in the time frame going from fault detection to when fault recovery has been completed.

We propose a classification of the fault affecting a portion of the FPGA based on the *absolute and relative frequency* of the detected faults. More precisely, the defined strategy keeps track of both the number of faults occurring in each separate recoverable area, IRAs and controller, and of their history, recording the locations of the most recent observed faults. The rationale is that, if an area is affected by a fault more frequently than the others, a relocation should take place to prevent further use of the area. To support this strategy, the module has $n+1$ separate counters, to count the number of faults per

area, and a *trend buffer* to track the most recent areas affected by the faults; the dimension of the buffer is determined by the application scenario, in such a way that the higher the expected SEU frequency, the deeper the buffer. It is worth noting that the adopted methodology for system hardening produces solutions with a limited number of IRAs (see [8] for more details), thus the number of counters is small and the buffer has a manageable size. The foreseen system lifetime is used to estimate a k threshold, together with the SEU incidence. A combination of absolute and relative frequencies of the area corruption contributes to determine which kind of reconfiguration to apply. More in detail, the fault classification process considers i) the faulty area presence in the buffer, ii) the absolute value of the counter related to the faulty area with respect to k, and iii) the relative value of the counter with respect to the other counters, as reported in the flow diagram in Figure 5.

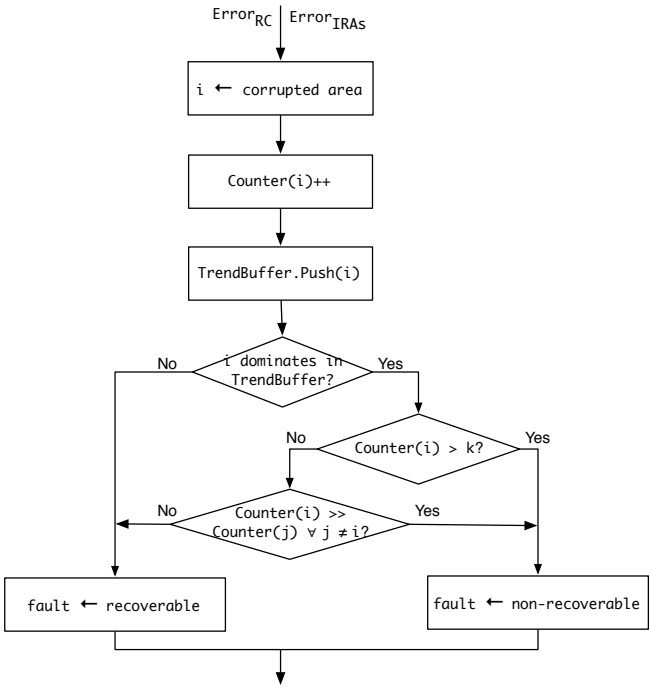

Figure 5. Fault classification flow diagram

The output of the Fault Classifier module feeds the Bitstream Module with the information about having detected or not a fault, and on the derived nature, recoverable or non-recoverable, to trigger a partial or complete reconfiguration of the neighbor FPGA. The output configurations are 01 (no error), 00 (recoverable fault) and 11 (non-recoverable fault).

C. Bitstream Module

Upon error detection and based on the fault classification, the Bitstream Module disables fault classification and triggers a reconfiguration action, accessing the protected memory to retrieve the appropriate configuration data. With such information, it programs the Reconfiguration Interface to carry out the FPGA (partial) reconfiguration to recover from the fault. The bitstreams are organized in memory as follows: the first bitstream consists of the current configuration of the monitored FPGA and is used for partial reconfiguration in case of a recoverable fault. The other bitstreams are used to recover from non-recoverable faults, each of them dealing with one faulty area.

To implement such functionality, inputs to the Bitstream Module are the `FaultType` signal, the $\texttt{Error}_{\texttt{IRA}}$, and $\texttt{Error}_{\texttt{RC}}$ signals. When necessary, the Bitstream Module interacts with the memory to retrieve the desired (partial) bitstream, `ConfigurationData`, to be forwarded to the Reconfiguration Interface, together with a `ReconfigurationRequest`, in terms of `FAR`, `Word`, and `Data` (i.e., the information necessary to perform the reconfiguration). An acknowledge signal, `Ack`, is used to increase the Reconfiguration Controller reliability, by having an additional flag to check that reconfiguration has been performed correctly. When the `Ack` is received it means that fault recovery is complete and fault classification can be re-enabled (through `Enable` signal), restoring the system to its nominal behavior.

V. Approach evaluation

A preliminary evaluation of the proposed architectural solution allows us to draw the following considerations describing the implementation choices.

Reconfiguration Controller implementation

Both software and hardware implementations of the Reconfiguration Controller have been taken into account, to identify the most convenient solution, based also on the available FPGA platform. A preliminary software implementation has been developed and deployed for a board hosting a microprocessor, to refine the controller behavior and to verify the external reconfiguration aspect, triggered by the error detection. Two are the main limitations characterizing a software implementation: sharing the microprocessor with the application being normally executed, and the hardening against faults. As far as the former aspect is concerned, should the available microprocessor be exploited to run the nominal application (otherwise the platform would not host a microprocessor, to begin with), the fault mitigation management application would have to be executed periodically, during idle time, thus introducing a latency in fault detection and management. Considering the latter aspect, the prototype did not expose reliability features, since hardening by means of software-based techniques is expensive in terms of overhead (code and performance degradation) as well as it only allows for a partially reliable system. A possible solution to achieve the necessary reliability consists in adopting a fault tolerant IP, such as Leon2-FT [21]; however, the final cost is quite prohibitive, with an occupation of available resources on an xc2v3000 FPGA requiring about 24% registers, 37% Block RAMs and 79% LUTs. For these reasons, a hardware solution, whose design is presented in the previous section, has been preferred, and is currently being developed and evaluated.

Relocation bitstream

Research for managing different configuration bitstreams for an FPGA can be broadly classified into two cathegories: i) *on-line* bitstream computation, that dynamically creates the alternatives configurations as in [22], and ii) *off-line* bitstream computation, that creates the alternatives configurations during the design phase as in [23]. 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 IRAs; the latter one, instead, consists of computing all the bitstreams that might be needed before deploying the system and, at runtime, only the correct bitstream is loaded. We preliminarily evaluated both strategies, to adopt the most convenient one for our scenario, where the different bitstreams do not correspond to different tasks/features to be loaded on the device, but to the same system being distributed on the reconfiguration fabric in another way. The former approach would make the Reconfiguration Controller heavily dependent on the actual FPGA and it would consistently increase its complexity; given that the number of alternatives configuration is usually quite low, our approach has been based on the off-line computation, pre-loading in memory all the bitstreams which might be needed during system lifetime.

Distributed control architecture

The adoption of a distributed fault mitigation manager allows the system to survive the degradation of an FPGA, with a more robust solution with respect to having an FPGA hosting a single master Reconfiguration Controller, which would become a particularly critical element, fragile to non-recoverable faults. In fact, the single controller solution represents a single point of failure, consequently requiring to be implemented onto a particular device (e.g. a radiation-hardened ASIC, as in [5]).

VI. Conclusions and future work

This paper proposes the design of reliable complex embedded systems spreading over multi-FPGA platforms. A reconfiguration-based strategy for managing both recoverable and non-recoverable faults has been proposed, focusing on the main component responsible for implementing such strategy, namely the Reconfiguration Controller, whose functionality and behavior have been described. The most suitable implementation of the Reconfiguration Controller has been identified by evaluating different architectural solution. The devised solution presents, with respect to the state of the art, various innovative aspects: i) distributed, low cost Reconfiguration Controller, eliminating the need to deploy it on ad-hoc device, ii) management of both recoverable and non-recoverable faults, and iii) scalable use of multiple FPGAs to build a reconfigurable reliable system.

In future work, the designed Reconfiguration Controller will be extended by i) characterizing the faults occurring on the device, ii) refining the fault classification algorithm, iii) evaluating analytically and quantitatively the architectural solution, and iv) considering the system behavior after a reconfiguration.

REFERENCES

[1] R. Roosta, "A comparison of radiation-hard and radiation-tolerant FPGAs for space applications," NASA - Jet Propulsion Laboratory, Tech. Rep. JPL D-31228, 2004.

[2] R. Katz, K. LaBel, J. J. Wang, B. Cronquist, R. Koga, S. Penzin, and G. Swift, "Radiation effectcs on current field programmable technologies," *IEEE Trans. Nuclear Science*, vol. 6, no. 44, pp. 1945–1956, 1997.

[3] S. D'Angelo, C. Metra, S. Pastore, A. Pogutz, and G. R. Sechi, "Fault-Tolerant Voting Mechanism and Recovery Scheme for TMR FPGA-Based Systems," in *Proc. IEEE Int. Symp. Defect and Fault-Tolerance in VLSI Systems*, 1998, pp. 233–240.

[4] F. L. Kastensmidt, G. Neuberger, R. F. Hentschke, L. Carro, and R. Reis, "Designing Fault-Tolerant Techniques for SRAM-Based FPGAs," *IEEE Design and Test of Computers*, vol. 21, no. 6, pp. 552–562, 2004.

[5] G. L. Smith and L. de la Torre, "Techniques to enable FPGA based reconfigurable fault tolerant space computing," in *Proc. IEEE Aerospace Conference*, 2006.

[6] V. Rana, M. Santambrogio, D. Sciuto, B. Kettelhoit, M. Köster, M. Porrmann, and U. Rückert, "Partial Dynamic Reconfiguration in a Multi-FPGA Clustered Architecture Based on Linux," in *Proc. IEEE Int. Parallel and Distributed Processing Symp.*, 2007, pp. 1–8.

[7] P. K. Samudrala, J. Ramos, and S. Katkoori, "Selective triple Modular redundancy (STMR) based single-event upset (SEU) tolerant synthesis for FPGAs," *IEEE Trans. Nuclear Science*, vol. 51, no. 5, pp. 2957–2969, Oct. 2004.

[8] C. Bolchini and A. Miele, "Design Space Exploration for the Design of Reliable SRAM-based FPGA Systems," in *Proc. IEEE Int. Symp. Defect and Fault-Tolerance in VLSI Systems*, 2008, pp. 332–340.

[9] C. Carmichael, M. Caffrey, and A. Salazar, "Correcting Single-Event Upsets Through Virtex Partial Configuration," Xilinx Inc., Tech. Rep. XAPP216, June 2000.

[10] C. Claus, B. Zhang, W. Stechele, L. Braun, M. Hübner, and J. Becker, "A multi-platform controller allowing for maximum Dynamic Partial Reconfiguration throughput," in *Proc. Int. Conf. Field Programmable Logic and Applications*, 2008, pp. 535–538.

[11] C. Claus, F. H. Muller, J. Zeppenfeld, and W. Stechele, "A new framework to accelerate Virtex-II Pro dynamic partial self-reconfiguration," in *Proc. IEEE Int. Parallel and Distributed Processing Symp.*, 2007, pp. 1–7.

[12] A. Cuoccio, P. R. Grassi, V. Rana, M. D. Santambrogio, and D. Sciuto, "A Generation Flow for Self-Reconfiguration Controllers Customization," in *IEEE Int. Workshop Electronic Design, Test and Applications*, 2008, pp. 279–284.

[13] S. Bayar and A. Yurdakul, "Dynamic Partial Self-Reconfiguration on Spartan-III FPGAs via a Parallel Configuration Access Port (PCAP)," in *Proc. HiPEAC Workshop on Reconfigurable Computing*, 2008.

[14] Xilinx, "Virtex-II Pro and Virtex-II Pro X FPGA User Guide," Tech. Rep. UG012, November 2007.

[15] ——, "Virtex-4 FPGA Configuration User Guide," Tech. Rep. UG071, June 2009.

[16] ——, "OPB HWICAP Product Specification," Tech. Rep. DS280, March 2004.

[17] E. Normand, "Single Event Upset at Ground Level," *IEEE Trans. on Nuclear Science*, vol. 43, no. 6/1, pp. 2742–2750, 1996.

[18] S. Srinivasan, P. Mangalagiri, Y. Xie, N. Vijaykrishnan, and K. Sarpatwari, "FLAW: FPGA lifetime awareness," in *Proc. ACM Design Automation Conf.*, 2006, pp. 630–635.

[19] D. Nikolos, "Self-testing embedded two-rail checkers," *J. Electronic Testing, Theory and Applications*, vol. 12, no. 1-2, pp. 69–79, 1998.

[20] Xilinx, "Virtex FPGA Series Configuration and Readback," Tech. Rep. XAPP138, March 2005.

[21] E. Microelectronics, "LEON2-FT," Website, 2009, http://www.esa.int/TEC/Microelectronics/SEMUD70CYTE_0.html.

[22] S. Corbetta, F. Ferrandi, M. Morandi, M. Novati, M. Santambrogio, and D. Sciuto, "Two Novel Approaches to Online Partial Bitstream Relocation in a Dynamically Reconfigurable System," in *Proc. IEEE Computer Society Annual Symp. VLSI*, 2007, pp. 457–458.

[23] W. Huang, S. Mitra, and E. McCluskey, "Fast Run-Time Fault Location in Dependable FPGA-Based Applications," in *Proc. IEEE Int. Symp. Defect and Fault Tolerance in VLSI Systems*, 2001, pp. 206–214.