A novel methodology to increase fault tolerance in autonomous FPGA-based systems

A novel methodology to increase fault tolerance in autonomous FPGA-based systems

Stefano Di Carlo, Giulio Gambardella, Paolo Prinetto, Daniele Rolfo, Pascal Trotta, Alessandro Vallero
Politecnico di Torino
Dipartimento di Automatica e Informatica
Corso Duca degli Abruzzi 24, I-10129, Torino, Italy
Email: {name.familyname}@polito.it
Telephone: (+39) 011.090-7191

***Abstract*—Nowadays, Field-Programmable Gate Arrays (FPGAs) are increasingly used in critical applications. In these scenarios fault tolerance techniques are needed to increase system dependability and lifetime.**
This paper proposes a novel methodology to achieve autonomous fault tolerance in FPGA-based systems affected by permanent faults. A design flow is defined to help designers to build a system with increased lifetime and availability. The methodology exploits Dynamic Partial Reconfiguration (DPR) to relocate at run-time faulty modules implemented onto the FPGA. A partitioning method is also presented to provide a solution which maximizes the number of permanent faults the system can tolerate.
Experimental results highlight the negligible performance degradation introduced by applying the proposed methodology, and the improvements with respect to state-of-the-art solutions.

I. Introduction

Field Programmable Gate Arrays (FPGAs) are increasingly used in many application fields. Nowadays, FPGAs are commonly adopted not just for Application Specific Integrated Circuits (ASICs) prototyping, but also for the implementation of embedded systems in critical scenarios (e.g., automotive or aerospace [1][2]) that require high reliability, availability, and long system lifetime. They offer limited Non-Recurrent Engineering (NRE) costs, and high flexibility, thanks to their dynamically reconfigurable characteristics.
FPGA dynamic reconfiguration can be effectively exploited to allow remote, or on-site, run-time system maintenance. In fact, when the system availability is a priority, remote repairing/maintenance is not the best solution, since it is slow and not always possible. In this context, systems able to autonomously self-recover are preferred. These systems are called Autonomous Fault-Tolerant Systems (AFTSs). They are of great interest because they offer increased lifetime and availability [3].
SRAM-based FPGAs are subject to two kinds of errors: soft-errors and hard-errors. In general, soft-errors (such as Single Event Upsets (SEUs) and Multiple Bit Upsets (MBUs) [4][5]) are induced by radiations and are temporary. On the other hand, hard-errors are caused by permanent faults and they are induced by device wear-out and aging [6]. In the last years many solutions investigated how to deal with soft-errors [7][8] while just few works were presented to cope with hard-errors [9][10]. Filling this gap is gaining importance as it has been remarked by the International Road Map for Semiconductors [11]. Permanent faults occur more frequently in modern systems as devices and wires have increasingly smaller dimensions and they operate at high temperatures. Furthermore, in those scenarios in which devices are expected to last several years, system lifetime is one of the most relevant aspects of the design. Therefore, aging and wear-out effects must be taken into account.
This paper proposes a design methodology to achieve autonomous fault tolerance in FPGA-based systems affected by permanent faults. A design flow is defined to help designers to build a system with increased lifetime and availability. The methodology exploits the Dynamic Partial Reconfiguration (DPR) feature of modern FPGAs [12] to relocate at run-time faulty modules implemented onto the device. DPR allows fast self-recovery from permanent faults, guaranteeing high availability and increased system lifetime. A parition-ing method is also presented to provide a solution which maximizes the number of permanent faults the system can tolerate. The proposed design methodology improves existing literature solutions by dramatically reducing recovery time and memory space needed to store recovery information. The rest of the paper is organized as follows. Section II briefly overviews existing literature solutions to cope with permanent faults on FPGA-based systems, highlighting the limitations they suffer from. Section III introduces the proposed paritioning and recovery methodology, while, Section IV reports results gathered from two case studies where the proposed methodology is adopted. Eventually, Section V concludes the paper and discusses possible future works.

II. Related Works

In literature, many solutions have been proposed to detect permanent faults in FPGA-based systems [13] [14], while, only few works address the fault tolerance improvement, or the recovery of this kind of systems from such faults.
To increase the fault tolerance of FPGA-based systems two alternative approaches can be exploited. The former exploits the introduction of redundancy, at design-time, allowing system operations without any interruption even in presence of faults. An example of such technique is the Triple Modular Redundancy (TMR), which consists of triplicating the hardware functionality to detect faults which cause errors in the outputs of a module. Nevertheless, this technique incurs in large hardware resources overhead.
The second, more efficient, approach consists of exploiting the dynamically reconfigurable capability of modern FPGAs to recovery from permanent faults, enabling to obtain Autonomous Fault-Tolerant Systems (AFTS) [3]. The idea of

exploiting dynamic reconfiguration to cope with permanent faults affecting a system implemented on a SRAM-based FPGA is not new [15][16][17][18].
The main idea is that, when a permanent fault caused by electromigration or device aging [6] affects a portion of the FPGA fabric, making it unusable, a different circuit configuration needs to be loaded, avoiding the usage of the permanently damaged area. This recovery action requires that the corrupted area is identified and delimited.
The most common solution to this problem is the so called *relocation*: whenever a portion of the FPGA is detected as faulty, the function implemented by that portion is moved to a spare reconfigurable area. The process is usually performed without stopping the portions of the FPGA that are not involved in the relocation so that other functionalities of the system remain available. Relocation consists of loading the proper configuration file, called *bitstream*, into the FPGA configuration memory. In particular, it is possible to configure the whole device [9][19] or, exploiting Dynamic Partial Reconfiguration (DPR) [12], to configure just a part of it [10].
In [9] and [19], authors present a detection and recovery methodology targeting both transient and permanent faults. Whenever a permanent fault is detected, the proposed recovery strategy consists of reconfiguring the whole FPGA with a precomputed different bitstream so that the circuit will not use the faulty FPGA resource. However, the number of full bitstream to store increases exponentially with the number of portions in which the design is split in. In this case, large recovery time is expected, since the full FPGA configuration bitstream is composed of several Mbits of data. Moreover, large memories storing all possible bitstreams are needed.
Instead, in [10] authors propose a DPR-based methodology for pipelined circuits. The usage of FPGA partial reconfiguration slightly reduces the recovery time and the memory requirements for bitstreams storing. However, the proposed methodology is not efficient, since even if a single circuit module is targeted as faulty, all the following modules must be also reconfigured. Authors do not address the problem of faults affecting the interconnections among adjacent modules, making the proposed methodology not applicable in real use-cases.
The methodology presented in this paper overcomes the aforementioned limitations exploiting Dynamic Partial Reconfiguration to relocate, at run-time, faulty modules implemented on the FPGA. Fast recovery time is guaranteed since it requires only the faulty module relocation, and the interconnection network update. Faults affecting interconnections between modules are taken into account by introducing spare resources reserved for the recovery of faulty interconnections. Moreover, the proposed methodology requires to store partial bitstreams whose size is dramatically reduced with respect to full FPGA configuration bitstreams.
Finally, an optimal paritioning method is presented to maximize the faults the system can tolerate, considering the free resources in FPGA device. The presented methodology enables to increase the availability and the lifetime of FPGA-based Autonomous Fault-Tolerant Systems.

III. Proposed methodology

Basically, the proposed recovery strategy consists of run-time relocation of faulty modules in spare FPGA resources. To support this strategy, an FPGA-based architecture and a paritioning methodology are defined. Run-time relocation is obtained by means of Dynamic Partial Reconfiguration (DPR) [12].model is designed to find a paritioning able to maximize the number of permanent fault the system can tolerate.

A. Proposed architecture

The proposed architecture addresses to the implementation of AFTSs on SRAM-based FPGAs. The designed systems must be able to autonomously detect and recover from faults. For this reason the architecture is composed of three blocks, each of them implementing different functions. As illustrated in Fig. 1, the three main components are:

- an SRAM-based FPGA hosting the hardware functionality, called *Application FPGA*,
- a *Fault Manager* which contains the *Configuration Controller* and the *Fault Classifier*,
- a *Bitstream memory* storing the *Application FPGA* configuration files.

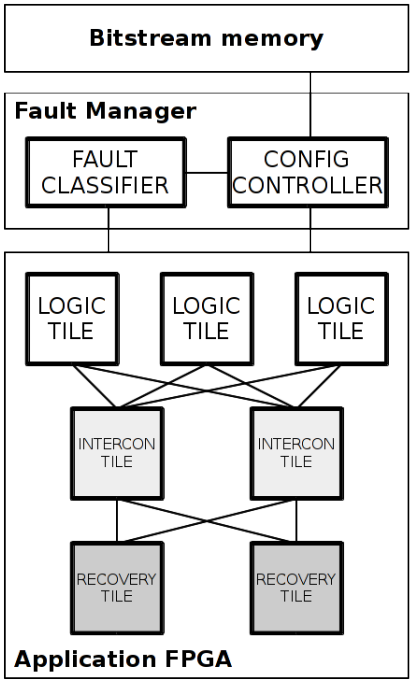


Figure 1: Proposed system architecture

The *Fault Manager* monitors faults that occur in the application FPGA and manages the recovery process. In details, the *Fault Classifier* detects faults and it establishes where they have happened. This task can be accomplished by running periodic tests on the *Application FPGA* [13][14], or simply collecting error signals generated by fault detection hardware embedded in the *Application FPGA* modules [20][21].
The *Configuration Controller*, instead, runs the recovery operations so that the system is restored back to a working state. It is important to notice that recovery operations are managed on the basis of the *Fault Classifier* diagnosis. For an SRAM-based FPGA system, recovery from permanent faults

means a reconfiguration of the FPGA. As a consequence, the *Configuration Controller* is connected to the *Bitstream memory*, so that a direct access to configurations is always guaranteed.
As robustness is of primary importance, both the *Configuration Controller* and the *Fault Classifier* can be implemented exploiting fault tolerance design techniques [22][23]. Moreover, the *Bitstream memory* can implement error detection and correction codes to avoid errors while the configuration files are read [24]. However, the actual *Fault Manager* and the *Bitstream memory* implementations are out of the scope of this paper.
The *Application FPGA* hosts the system's hardware functionalities. It is divided into several partitions, called *tiles*. The characteristics of *tiles* and their paritioning methodology are discussed in the following subsection.

B. Paritioning methodology

The *Application FPGA* provides three kinds of *tiles*, depending on their employment in the final system (Fig. 1). *Logic tiles* host the circuits that perform computation and data processing. *Recovery tiles* are adopted as spare tiles. When a permanent error occurs in a *logic tile*, the functionality implemented by the faulty tile is relocated into a *recovery tile*. Finally, *interconnection tiles* host the wires allowing communication among tiles.
During the design phase, the hardware functionality is built with a modular approach. The whole circuit is divided into basic components, which are interconnected to each other. Each basic component is then characterized by a certain amount of required FPGA resources (i.e., Slices, BRAMs and DSPs [25]) and by the number of connections with the other basic components.
Basic components are grouped and organized in logic tiles (Fig. 2). To host basic components, a logic tile must satisfy their resource requirements. As a result, logic tiles are characterized on the basis of the amount of resources and interconnections of the hosted components.

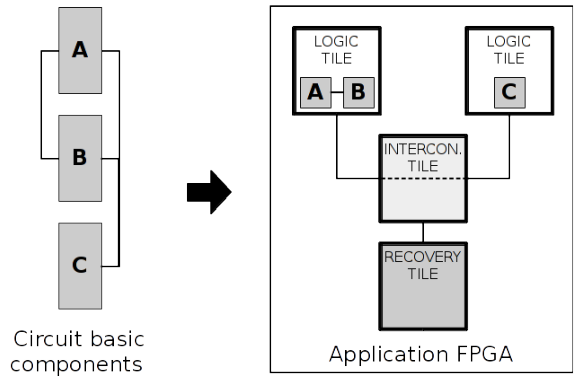
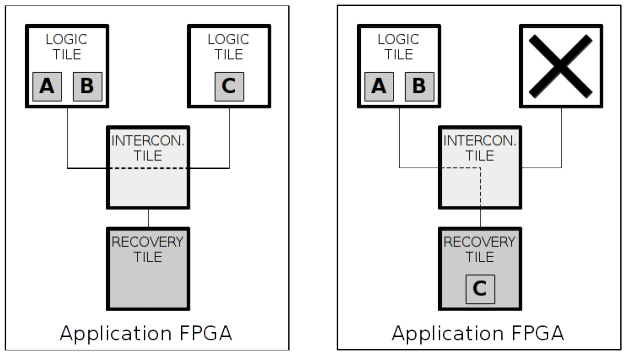


Figure 2: Circuit basic components and tiles organization

When a logic tile needs to be moved to a recovery tile because of a permanent fault, it must be guaranteed that the recovery tile has a sufficient number of resources.
All recovery tiles are sized in order to provide a number of resources equal to the one required by the most demanding logic tile. As a result, each logic tile can be moved to each recovery tile. Consequently, the number of recovery tiles is equal to the maximum number of faulty logic tiles the system can tolerate.
When a permanent fault in a logic tile is detected, the *Configuration Controller* loads the proper configuration file into the FPGA configuration memory to relocate the faulty tile into a recovery tile (Fig. 3a and 3b). However, this operation is not sufficient as the interconnections have to be updated too. To overcome this problem, the active interconnection tile is also reconfigured by the *Configuration Controller*, so a new configuration file for the interconnections tile is loaded (Fig.3b).
Backup interconnection tiles are added to the design to cope with faults affecting interconnections (Fig.3c). However, there is just an active interconnection tile at the time, while the remaining ones are employed only when faults occur. The number of required backup interconnection tiles must be equal to the number of interconnection faults (i.e., faults affecting interconnection tiles) the system must tolerate. Whenever a permanent fault occurs in the active interconnection tile, a backup interconnection tile is activated, while the faulty one is no longer used (Fig.3d), and it is reconfigured with an empty partial bitstream (i.e., a bitstream which not contains any circuit information). The proposed architecture offers great flexibility as it does not rely on a fixed interconnection architecture among system modules (i.e., it can be applied to systems based on buses, point-to-point connections, interconnection networks, etc.).
To implement our methodology the following number of configuration files, *n_conf_files* are required:

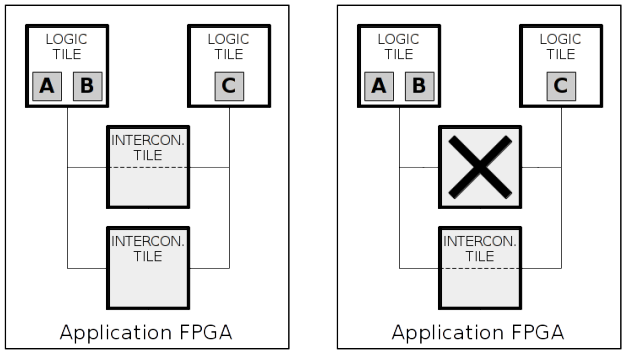
$$n\_conf\_files = n\_logic\_tiles \times n\_faults + \\ + \prod_{i=0}^{n\_faults-1} (n\_logic\_tiles - i) \times (n\_faults + 1) + \\ + n\_logic\_tiles \quad (1)$$

where *n_logic_tiles* is the number of logic tiles and *n_faults* is the number of permanent faults the system can tolerate. The first contribution is due to relocation of functions implemented in logic tiles to recovery tiles. The second contribution is due to interconnection tiles. In fact there are *n_faults + 1* interconnection tiles that must provide connection for all the possible combinations of logic tiles relocated to recovery tiles. Finally, the third term of Eq. 1 takes into account that faulty logic tiles must be reconfigured with an empty bitstream. The proposed recovery methodology provides two main improvements with respect to [9]. The first one concerns the recovery time. In fact, to recover from a permanent fault affecting a tile, partial configuration files have to be loaded, instead of reconfiguring the entire FPGA. Secondly, the memory required to store configuration files is dramatically reduced as we need a certain number of configuration files for interconnection tiles whose size is greatly reduced with respect to the ones required for the whole FPGA.
For the presented recovery strategy, the way the system is partitioned is extremely important. In fact, it influences the maximum number of permanent faults the system can tolerate. In fact, the largest number of recovery tiles that can be accommodated into the applications FPGA depends directly on how basic components are grouped and distributed among



(a) The system without any faulty logic tiles

(b) The system recovered from a faulty logic tile



(c) The system without any faulty interconnection tiles

(d) The system recovered from a faulty interconnection tile

Figure 3: The recovery strategies when a permanent faults occur. Fig 3a illustrates the relocation of the faulty logic tile into a recovery tile and the reconfiguration of the interconnection tile. Fig 3d shows how interconnections are reconfigured when a permanent fault is detected inside the active interconnection tile

logic tiles. The following subsection details the partitioning algorithm that can be used to find the basic components paritioning that maximize the number of tolerable faulty tiles.

C. Partitioning algorithm

Recovery tiles contain the necessary resources to accommodate every logic tile and their size is proportional to the number of slices, BRAM and DSP they offer. A fine-grained partitioning approach (e.g., assigning a basic component to each logic tile) leads to smaller recovery tiles, while a coarse-grained partitioning (i.e., grouping more than one component in a logic tile) requires larger tiles. In addition, changing partitioning means changing the number of wires inside the interconnection tiles, and so their area.

Because of the relevance of the partitioning strategy, we propose an algorithm to find a feasible paritioning offering the maximum number of faulty tiles the system can tolerate (see Algorithm 1).

As explained in the previous subsections, a model of the circuit is obtained by characterizing all the logic tiles by a number

Algorithm 1 Partitioning algorithm

```
Const N_components                    ▷ # of basic components
Const FPGA_res                        ▷ resources of the application FPGA
max_tolerated_faults = 0;
for n_logic_tiles= 1 to N_components do
    for each possible partitioning composed of n_logic_tiles do
        Slack_res = FPGA_res − Logic_tiles_res − Interc_tile_res
        n_tolerated_faults = Slack_res / (Rec_tiles_res+Interc_tile_res)
        if n_tolerated_faults > max_tolerated_faults then
            max_tolerated_faults = n_tolerated_faults;
            update best partitioning;
        end if
    end for
end for
```

of resources and connections they require. The resources demanded by each logic tile depends on the basic component circuits it accommodates, while the number of connections is related to the partitioning. It is important to notice that interconnection tiles do not require any resource since they do not perform computation, instead they just need connections. As a consequence, the delay introduced by interconnection tile is assumed negligible for the most of the applications (as will be shown in Sec. IV) since the critical path is bounded in a basic component.

In Algorithm 1, *Rec_tiles_res* represents the resources needed by a single recovery tile, *Interc_tile_res* is an amount of FPGA slices containing interconnections among all the logic and recovery tiles, while *Slack_res* are the spare resources when only logic tiles and one interconnection tile are taken into account.

Starting from the resources available on the target FPGA and the resources required by every basic component, the algorithm finds the best partitioning solution in terms of tolerable faulty tiles. Basic component circuits are partitioned with different granularities. The number of logic tiles ranges from one, a single huge partition which represents the coarsest granularity, to the number of basic components defined in the modular design, the finest granularity. For each iteration all possible basic components grouping combinations are analyzed, and the number of tolerated faulty tiles is computed. To compute the number of tolerated faulty tiles for a given partitioning it is supposed that for each recoverable permanent fault there is one recovery tile and one backup interconnection tile.

As aforementioned, interconnection tiles do not require any computational element. Nevertheless, because of the technology imposed by FPGA vendor, reconfigurable partitions require input/output pins, represented by FPGA Look-Up Tables [12].

The resulting number of tolerated faults, *n_tolerated_faults*, is the lowest among the ones computed for every type of resource. When *n_tolerated_faults*is greater than the temporary maximum number of tolerated faults, *max_tolerated_faults*, it is saved as the best partitioning.

The proposed algorithm is complex from a computational point of view, that is, it scales badly with the increasing number of *n_logic_tiles*. However, since it must be executed just once at design-time, its execution time does not influence the overall performance of the implemented system.

IV. Experimental Results

To analyze and measure the performancec of the proposed methodology, it has been applied to two case studies.
A Xilinx *Virtex-4 VSX55* FPGA has been chosen as the target *Application FPGA* as it provides a large number of slices, BRAMs and DSPs [26]. This FPGA can be dynamically and partially reconfigured by means of the *SelectMAP* port at a maximum reconfiguration throughput equal to 400MB/s [12]. For both case studies, basic components are characterized by their resource requirements and these values are fed into the proposed partitioning algorithm. The partitioning that maximizes the number of tolerated faulty tiles is chosen for the final system implementation. Performance are analyzed in terms of number of tolerated permanent faulty tiles, system recovery time and bitstream size.
In the first case study the proposed methodology is applied on an IP-core for images feature extraction and matching, called *FEMIP* [27]. As presented in Fig. 4, *FEMIP* is composed of five basic components. In Fig. 4, numbers between components

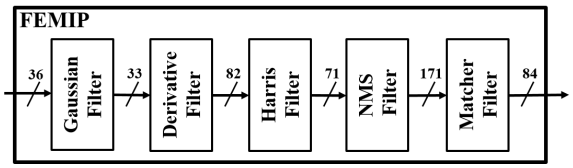


Figure 4: FEMIP basic components

represent the interconnection widths. Resources required by each component are reported in Table I.

Table I: Resources requirements for *FEMIP* basic components

Component	# slices	# BRAMs
Gaussian filter	2560	8
Derivative filter	1344	8
Harris filter	3456	0
NMS filter	360	4
Matcher filter	650	6

Taking into account the overall resources available in the target *Application FPGA*, the maximum number of tolerated faulty tiles, obtained with the proposed partitioning algorithm, is four. Resources requirements for the best partitioning are shown in Table II. The number of recovery tiles and backup

Table II: Resources requirements for FEMIP tiles

Tile	Components	# slices	# BRAMs	Bitstream
Logic 1	Gaussian filter	2560	8	254.5 KB
Logic 2	Derivative filter	1344	8	144.6 KB
Logic 3	Harris filter	3456	0	360.1 KB
Logic 4	NMS filter Matcher filter	1010	10	120.0 KB
Interconnection (x5)		128	-	7.2 KB
Recovery (x4)		3456	10	360.1 KB

interconnection tiles is four. Exploiting Eq. 1, the number of partial bitstreams to store in the *Bitstream Memory* is equal to 72. Thus, the memory required is $4,192KB$, $2,880.8KB$ for recovery tiles (8 configuration files), $432KB$ for interconnection tiles (60 configuration files) and $879.2KB$ for empty logic tiles (4 configuration files in total). As reconfiguration throughput is 400MB/s, the worst case recovery time is equal to $1.82ms$ (including the reconfigurations of the faulty logic tile with the empty bitstream, the relocated recovery tile, and the interconnection tile).
The introduction of the interconnection tile to allow communications between tiles does not affect timing performance of the system for two reasons. First, the critical path is bounded inside basic components logic. Secondly, the delay introduced by interconnection tiles is $0.2ns$, which is negligible with respect to the *FEMIP* minimum clock period. As a consequence, the maximum operating frequency of *FEMIP* remains equal to the one obtained without any fault tolerance technique (i.e., $60MHz$).
To allow comparison with respect to the methodology presented in [9], the proposed methodology has been applied to an H.246 video encoder. The comparison has been made in terms of number of tolerated faults, bitstreams size, and recovery time. Basic components and relative resources requirements for the H.246 video encoder can be found in [9].
Performance of the partitioning selected by our algorithm are reported in Table III. With the given partitioning it is

Table III: Resources requirements for H.264 video encoder tiles

Tile	Components	# slices	# BRAMs	# DSPs	Bitstream
Logic 1	intra8x8cc coretransform intra4x4 recon	3225	6	0	349.3 KB
Logic 2	dctransform calvc	2811	0	0	296.5 KB
Logic 3	buffer process1 quantise invdctransform dequantise invtransform	3123	6	6	380.3 KB
Logic 4	process2	3120	0	0	349.3 KB
Logic 5	header tobyte	1605	0	0	176.5 KB
Interconnection (x3)		320	0	0	18.0 KB
Recovery (x2)		3225	6	6	380.3 KB

possible to recover from two permanent faults, as in [9]. For this purpose two recovery tiles and backup interconnection tiles are accommodated in the *Application FPGA*. For our implementation the total amount of required *Bitstream memory* is $6434.9KB$ (75 configuration files in total), $3803KB$ for the recovery tiles (10 configuration files in total), $1080KB$ for the interconnection tiles (60 configuration files in total) and $879.2KB$ for empty logic tiles (5 configuration files in total). Although the number of bitstream is comparable to the one of [9], the memory space required for bitstreams is dramatically reduced. Thanks to dynamic partial reconfiguration, it is almost 35x smaller than $219MB$ demanded for a recovery strategy consisting in reconfiguring the entire *Application FPGA*.
Finally, the proposed methodology allows to shorten the recov-

ery time since just a part of the implemented circuit is to be configured. In fact, in the worst case, $1.95ms$ are required to recover from a permanent fault, leading to a 4x improvement with respect to the $7.5ms$ needed when a full reconfiguration of the FPGA is performed (as in [9]).

V. Conclusions

This paper presented a novel methodology to increase availability and lifetime of FPGA-based systems affected by permanent faults. The methodology exploits Dynamic Partial Reconfiguration (DPR) to relocate at run-time faulty modules. A partitioning method is also presented to provide a solution which maximizes the number of permanent faulty modules the system can tolerate.
Experimental results highlight the negligible performance degradation introduced by applying the proposed methodology, and the improvements in terms of both fault recovery time and memory requirements with respect to state-of-the-art solutions. Future works will focus on increasing the relocation efficiency, in order to further decrease the recovery time, and on the development of a framework, which includes FPGA device models, to allow designer to accurately and automatically apply the proposed methodology. In addition, some efforts will address the optimization of the partitioning algorithm to reduce its complexity, e.g., by means of heuristic techniques.

References

[1] Xilinx Corporation, *Automotive Driver Assistance Systems: Using the Processing Power of FPGAs - White paper (WP399)*, 2011.

[2] A. Hofmann, R. Wansch, R. Glein, and B. Kollmannthaler, "An FPGA based on-board processor platform for space application," in *Adaptive Hardware and Systems (AHS), 2012 NASA/ESA Conference on*, pp. 17–22, June 2012.

[3] X. Iturbe, K. Benkrid, T. Arslan, I. Martinez, M. Azkarate, and M. Santambrogio, "A roadmap for autonomous fault-tolerant systems," in *Design and Architectures for Signal and Image Processing (DASIP), 2010 Conference on*, pp. 311–321, Oct 2010.

[4] A. Lesea, S. Drimer, J. J. Fabula, C. Carmichael, and P. Alfke, "The rosetta experiment: atmospheric soft error rate testing in differing technology FPGAs," *Device and Materials Reliability, IEEE Transactions on*, vol. 5, no. 3, pp. 317–328, 2005.

[5] Xilinx Corporation, *Continuing Experiments of Atmospheric Neutron Effects on Deep Submicron Integrated Circuits - White paper (WP286)*, 2011.

[6] N. Mehta and A. DeHon, "Variation and aging tolerance in fpgas," in *Low-Power Variation-Tolerant Design in Nanometer Silicon*, pp. 365–380, Springer, 2011.

[7] C. Carmichael, E. Fuller, P. Blain, and M. Caffrey, "Seu mitigation techniques for virtex fpgas in space applications," in *Proceeding of the Military and Aerospace Programmable Logic Devices International Conference (MAPLD)*, p. C2, 1999.

[8] C. Bolchini, D. Quarta, and M. D. Santambrogio, "Seu mitigation for sram-based fpgas through dynamic partial reconfiguration," in *Proceedings of the 17th ACM Great Lakes symposium on VLSI*, pp. 55–60, ACM, 2007.

[9] C. Bolchini, A. Miele, and C. Sandionigi, "Autonomous fault-tolerant systems onto sram-based fpga platforms," *Journal of Electronic Testing*, vol. 29, no. 6, pp. 779–793, 2013.

[10] J. Zhang, Y. Guan, and C. Mao, "Optimal partial reconfiguration for permanent fault recovery on sram-based fpgas in space mission," *Advances in Mechanical Engineering*, vol. 2013, 2013.

[11] "ITRS: International technology roadmap for semiconductors - http://www.itrs.net/links/2011itrs/home2011.htm."

[12] Xilinx Corporation, *Partial Reconfiguration User Guide (UG702)*, 2013.

[13] L. Cassano, D. Cozzi, S. Korf, J. Hagemeyer, M. Porrmann, and L. Sterpone, "On-line testing of permanent radiation effects in reconfigurable systems," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2013*, pp. 717–720, March 2013.

[14] M. E. Imhof, M. A. Kochte, E. Schneider, H. Zhang, J. Henkel, and H.-J. Wunderlich, "Test strategies for reliable runtime reconfigurable architectures," *IEEE Transactions on Computers*, vol. 62, no. 8, 2013.

[15] Z. E. Rákossy, M. Hiromoto, H. Tsutsui, T. Sato, Y. Nakamura, and H. Ochi, "Hot-swapping architecture with back-biased testing for mitigation of permanent faults in functional unit array," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2013*, pp. 535–540, IEEE, 2013.

[16] S.-Y. Yu and E. J. McCluskey, "Permanent fault repair for fpgas with limited redundant area," in *Defect and Fault Tolerance in VLSI Systems, 2001. Proceedings. 2001 IEEE International Symposium on*, pp. 125–133, IEEE, 2001.

[17] S. Mitra, W.-J. Huang, N. R. Saxena, S.-Y. Yu, and E. J. McCluskey, "Reconfigurable architecture for autonomous self-repair," *IEEE Design & Test*, vol. 21, no. 3, pp. 228–240, 2004.

[18] J. Lach, W. H. Mangione-Smith, and M. Potkonjak, "Low overhead fault-tolerant fpga systems," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 6, no. 2, pp. 212–221, 1998.

[19] C. Bolchini, A. Miele, and C. Sandionigi, "Increasing autonomous fault-tolerant fpga-based systems' lifetime," in *Test Symposium (ETS), 2012 17th IEEE European*, pp. 1–6, May 2012.

[20] M. Violante, N. Battezzati, and L. Sterpone, "Reconfigurable field programmable gate arrays for mission-critical applications," 2011.

[21] F. G. d. L. Kastensmidt, G. Neuberger, R. F. Hentschke, L. Carro, and R. Reis, "Designing fault-tolerant techniques for sram-based fpgas," *IEEE Design & Test*, vol. 21, no. 6, pp. 552–562, 2004.

[22] J. Heiner, N. Collins, and M. Wirthlin, "Fault tolerant icap controller for high-reliable internal scrubbing," in *Aerospace Conference, 2008 IEEE*, pp. 1–10, IEEE, 2008.

[23] A. Ebrahim, K. Benkrid, X. Iturbe, and C. Hong, "A novel high-performance fault-tolerant icap controller," in *Adaptive Hardware and Systems (AHS), 2012 NASA/ESA Conference on*, pp. 259–263, IEEE, 2012.

[24] R. Micheloni, A. Marelli, and R. Ravasio, *Error correction codes for non-volatile memories*. Springer, 2008.

[25] Xilinx Corporation, *Virtex-4 FPGA User Guide - UG070*, 2008.

[26] Xilinx Corporation, *Virtex-4 FPGA Family Overview - DS112*, 2010.

[27] S. Di Carlo, G. Gambardella, P. Prinetto, D. Rolfo, P. Trotta, and P. Lanza, "Femip: A high performance fpga-based features extractor amp; matcher for space applications," in *Field Programmable Logic and Applications (FPL), 2013 23rd International Conference on*, pp. 1–4, Sept 2013.