An efficient BER-Based Reliability Method For SRAM-based FPGA

Fouad Sahraoui, Fakhreddine Ghaffari, Mohamed El Amine Benkhelifa, Bertrand Granado

An Efficient BER-based Reliability Method For SRAM-based FPGA

Fouad Sahraoui[1], Fakhreddine Ghaffari[1], Mohamed El Amine Benkhelifa[1], Bertrand Granado[2]

[1] ETIS, CNRS, ENSEA, Cergy-Pontoise University; 6 avenue du Ponceau, 95000 Cergy-Pontoise, FRANCE

[2] UPMC, LIP6, CNRS UMR 7606; 4 Place Jussieu, 75252 PARIS Cedex 05, FRANCE

[1]firstname.name@ensea.fr [2]firstname.name@lip6.fr

Abstract- Single Event Upset (SEU) is a major concern for SRAM-based FPGAs where a simple bit-flip can lead to an abnormal execution. We present in this paper, a new fault tolerance method based on hardware BER (Backward Error Recovery) to protect/correct system against the occurrence of transient faults. We use the partial dynamic reconfiguration offered by Xilinx Virtex-5 FPGAs to ensure hardware checkpoint and upon detection of fault we use recovery. Our method has several advantages: first it is non-intrusive (no internal modification of hardware modules of the system), second it is not based on redundant hardware resources (like most methods in the literature), and finally it has a static area overhead ratio when applied to a system. To validate our approach, we implemented it on a Xilinx platform based on a Partial Reconfigurable Region (PRR).

Index Terms- Fault tolerance, Checkpoint, Recovery, SRAM-based FPGAs, partial dynamic reconfiguration, readback.

I. INTRODUCTION

Modern FPGAs have reached a level of flexibility that can perform switching functionalities, on-line update and even environmental adaptability through partial dynamically reconfiguration; which is the ability to change the functionality of a portion of the chip without halting the rest of the FPGA [1, 2]. Furthermore, these FPGAs are no longer considered as only prototyping platform but also as execution platform, which lead to conduct new researches to enhance the execution of FPGA-based systems [3-5] through this ability.

To achieve this flexibility, SRAM-based FPGAs are composed of two distinct levels. A Configuration level**,** formed by a number of SRAM cells holding configuration data called *Bitstream*. This level controls the behavior of a Resources level, which is a distribution of basic elements: look-up-tables, flip-flops, multiplexers and dedicated blocks, all interconnected by configurable routing matrixes. While this type of platform joins between the flexibility of software implementation (processors for example) and high performance of application-specific integrated circuit (ASIC), it could introduce, depending on their running environment, an undesirable behavior at execution time. Radiation, ionization, extreme temperature variation and power fluctuation can affect the correct execution of FPGA-based system and lead to abnormal and erroneous results. A convenient solution for such harsh conditions is the use of radiation-hardened devices [6], where their internal structure offers a more reliable platform. Despite the improvement of reliability, these devices still lack a lot of features that are already available on SRAM-based FPGAs, like reconfiguration, high performance/density and low cost.

To benefit from the available SRAM-based FPGAs features, a number of fault tolerant techniques must be incorporated within the execution platform to enhance the overall reliability of the system. Among these methods, redundancy (Triple Modular Redundancy TMR [7], Duplication With Comparison DWC [8]) is one of the most simple and effective technique used in hardware to tolerate faults on FPGAs. Unfortunately these methods have some major drawbacks like: area/resources overhead, maximum frequency reduction, and huge increase of power consumption.

In this paper, we are seeking for an alternative approach for fault tolerance that is based on the mechanism of saving fault-free states of a running module and the mechanism of re-executing one of these saved states in case of an error detection. These two mechanisms are known as checkpoint and recovery. Formally, we investigate the feasibility to perform a checkpoint/recovery of hardware modules on FPGA platform. We validate this fault tolerant method on Xilinx FPGAs (Virtex-5 family) with respect to the following objectives.

- No modifications of internal structure of the module
- Take advantage of *partial dynamic reconfiguration* features

The rest of this paper is organized as follows. Section II covers faults tolerance methods proposed for SRAM-based FPGAs followed by the adopted fault model and the target device. Section III details each part of our backward error recovery and metrics definitions. An overview of the experimental platform based on a MicroBlaze is presented in Section IV. Finally, we conclude this paper and present some future works in Section V.

II. BACKGROUND AND RELATED WORK

A. Fault Tolerance Methods

Many investigations for fault mitigation on FPGAs have been conducted by researchers to enhance the reliability of SRAM-based FPGAs; a great number of them are based on the use of redundancy [9]. When redundancy is used, a

number of instances of the same module are added to the design and the results are compared by the mean of a voter to check the occurrence of faults. In the case of mismatched results the voter masks it and propagates the majority results. At the same time, the faulty instance can be corrected in the background if needed. One of the most known redundancy method in ASIC/FPGA is the *Triple Module Redundancy* (TMR) [7]. Such a method is formally proven to deal with any type of faults without any latency on one of these instances at the same time, but produces a heavy overhead in term of resource utilization (area), power consumption and speed diminution.

Some recent works were proposed to reduce the overhead generated by the TMR strategy. In [10], the authors proposed an online adaptive fault tolerant method relying on the partial reconfigurable region (PRR) feature and combined with different levels of redundancy. Depending on the level of space radiation and orbital position, a controller choses between five different modes of reliability to achieve better resources utilization and power saving. Sullivan et al. make a comparison between TMR and Reduced Precision Redundancy (RPR) in [11], and found that the RPR approach can be a practical alternative method for protecting arithmetic operations with an acceptable loss of precision on the result if a fault occurs.

Reconfiguration capability can be used alone to avoid redundancy in the form of *configuration scrubbing* [12], which involves a periodic complete or partial rewrite of initial bitstream to eliminate any eventual bit upset on the configuration layer [13]. This operation is performed by periodically retrieving the configuration data (bitstream) from a Golden memory (supposed to be less sensitive to faults compared to SRAM memory) and writes it via an access port to the configuration layer. Scrubbing method is merely based on statistical analyses of the execution environment, where the scrubber must always works at a rate higher than the one of faults occurrences. This method is adapted for environment with well-known rate of fault occurrences.

Information redundancy is another fault mitigation technique and can be a good choice to enhance the reliability of SRAM-based FPGAs. By using *Error Detection and Correction Codes* (EDAC codes) the configuration layer can be protected against possible bit inversion. *Extended hamming code* is a well-known EDAC code to protect memory-based system; but it can achieve only single error correction and double error detection (SECDED) [14]. An online single event upset controller is provided by Xilinx to detect and correct faults on the configuration memory and is detailed in [15]. The parity bits are calculated offline and stored in the configuration memory, later the controller will periodically check if an error occurred. Despite the fact that this controller is quite simple and has a small area overhead, it can be harmful for the protected system because if an odd number of faults occur and are greater than two then hamming syndrome will state that only one fault has occurred and will calculate a wrong bit position which leads to the introduction of more faults on the system. In [16] authors present a fault injection campaign and results that confirm this problem and point out that the system still contains at the end of injection a significant number of configuration bit errors.

Checkpoint/Recovery is an effective methodology for faults mitigation in processor based computing systems; the mechanism is widely used to protect scientific computation, critical applications and distributed systems from potential loss of calculations and system failure for many years. *Checkpoint* is the operation of saving all the necessary information of an application, called context, so it can be restarted later from that moment, and *Recovery* is the opposite operation where this information are written back to the executive platform, so the application can resume its computation from a fault-free state.

When it comes to FPGA based systems, this concept may at first seem to be inappropriate for such platform and furthermore not supported natively. The few studies we have found working with hardware checkpoint/recovery involve a deep transformation of the target module. In [17], three variants of hardware checkpoint are presented, each one introduces additional circuitries to retrieve the context of the hardware module. A tool was proposed to automatically transform the hardware module and introduce the additional access logics to the state of the module. These modifications are based on a set of netlist primitives substitution performed automatically. The main drawback is the alteration of the hardware module which can introduce design faults and performances degradation. Another work, presented in [18], investigates the integration of software and hardware checkpoint on a FPGA based cluster application. An interface to each hardware accelerator core (hardware module) is added to allow the access to the state of internal resources and by the way support the hardware checkpoint; this work is also based on an automatic transformation approach but applied at the VHDL level, which imposes certain constraints on the source code (must be described like an FSM) and a specific interface.

From our point of view, a modification approach is not always feasible or can induce a long time to apply on an existing system (even if it is automatically applied). Because of this, we explore the possibility of using actual FPGAs capabilities which provide partial dynamic reconfiguration to construct a checkpoint with no modification of the modules (a little modification/addition between the system's modules can be tolerated).

B. Fault Model

Faults are classified into two main categories: *permanent faults* and *transient faults* [19]. The first category includes the ones which produce a persistent erroneous behavior of the system. The only way to correct the affected part is to perform a system reboot or in extreme case a replacement of erroneous part with another correct instance. This type of fault is mostly induced by physical damage of the internal elements and leads to the alteration of elements, like for example a flip-flop which

stuck at a same state. Also, a long period of usage can leads to the same consequences. In all case a physical intervention is needed and shutdown of the system must be performed. On the contrary, the second category regroups faults which generate an erroneous behavior for a brief interval of time and disappear. If this type of fault hits the operative layer, a simple re-computation of the affected calculation is performed, while if it hits the configuration layer, a reconfiguration of the functionality is carried out to correct the system.

Mostly, SRAM-based FPGAs are more sensitive to transient faults than permanent ones. For example a simple ionized particle can hit the SRAM layer and leads to a bit-flip which can produce an erroneous behavior in the system at runtime. Another example of source for transient faults in SRAM is the voltage variation which could lead to abnormal operation of SRAM cells and introduction of faults.

The work presented here focuses on the correction of transient faults occurring at the SRAM layer of the FPGA and can lead to an erroneous behavior of the resource layer. Also we don't make distinction between a fault on a user data cell or on a configuration cell. We consider a fault as simple inversion of the initial value of a configuration cell on SRAM and it can propagate to data cells.

C. *Device model*

Resource layer in SRAM-based FPGAs is constructed by three main blocks which are: *logic blocks* formed by slices, *routing matrixes* to interconnect the logic blocks and *input/output blocks* to communicate with the external environment. Recent devices provide also embedded resources on the die, like Block RAM and DSPs, which aim to accelerate some specific part of hardware modules and do not need to be re-implement using logic blocks. Although the resources are heterogeneous; their arrangement on the fabric follows a homogenous distribution (with some exception for I/O), where each row contains the same number of resources columns and arranged identically from one row to the other. In Figure 1, we present an example of resources distribution of Xilinx FPGA to show this homogeneous distribution (we omitted the representation of routing for simplicity). This representation is valid from Virtex-4 up to virtex-7.

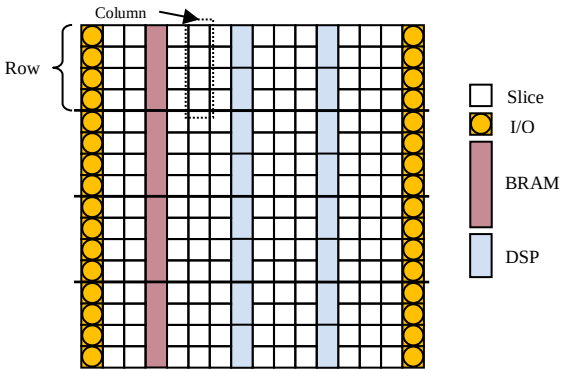

Figure 1. Example of resources repartition on Xilinx FPGA.

Each column of resource is configured with a specific number of *frames* stored on the configuration layer (SRAM). A frame is a vector of (1, n)-bits (n is specific to the device). From the point of view of configuration layer, frames are identical which mean that they have the same characteristics (height, width) but the interpretation of information contained inside each frame depends on the mapped resource to configure. For example in Virtex-5 [20], all frames have n=1312 bits height and 36 frames are needed to totally configure a column of logic blocks.

[20] provides an overview of the interpretation of frames configuring CLB and explanation of which part of frame maps a column of CLBs. Unfortunately, there is no details about the part configuring internal elements of CLB like for example LUTs. This information are still considered confidential, despite the existence of some works dealing with reverse/decode of Xilinx bitstream format [21, 22]. This is still not helpful for the purpose of our work. So we only take advantage of the information provided by Xilinx tools and documentations to localize and retrieve the context of a hardware module.

A CLB of Virtex-5 family is constituted of two slices, each one contains four 6-input LUTs, three multiplexers, a carry chain and four flip-flops. We identified that flip-flops are the location from which the context could be retrieved.

III. PROPOSED METHOD

In this section we are giving details of each part of our method which is based on features provided by the Xilinx FPGA (Virtex-5 [20]). A general overview of the proposed architecture is briefly represented in Figure 2. It aims to enhance the reliability of a user application which consists of a set of hardware modules placed in an *Enhanced Reliability Region (ERR)*. The ERR is protected with a *reliability controller* that implements detection, checkpoint and recovery mechanisms.

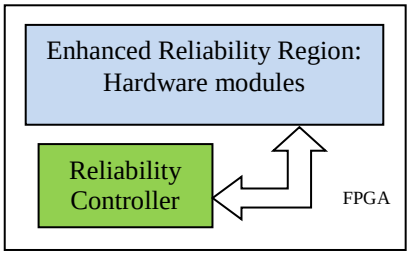

Figure 2. General representation of the proposed reliability architecture.

A. *Detection Strategy*

Our detection approach is performed based on the extended hamming code. Each frame of the FPGA contains parity bits calculated at bitstream generation step. Later at execution time, these bits are used to control the correctness of information contained in this frame. To prevent from the outlined problem of misdetection of odd number of faults (>2), the granularity of our detection method is set to a frame; it means that a detection of at least one fault in a frame makes

it completely faulty and induces a correction step of the module.

In figure 3, an overview of the structure of a frame configuring a CLB column is illustrated with the location of parity bits.

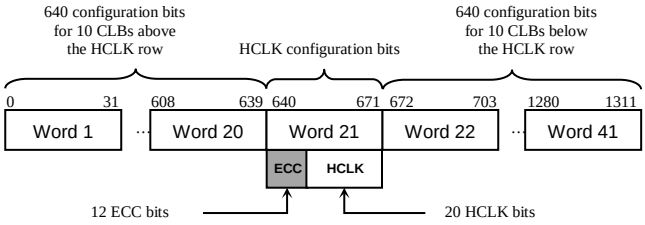

Figure 3. Configuration Bits in a Frame.

The detection strategy is conducted repeatedly on a set of frames configuring the enhanced reliability region, so errors can be identified at their early stage of occurrence and this minimizes their propagation. The worst scenario is where the last scanned frame is subject to a bit-flip. The time to detection in such scenario is given by the worst mean time to detection:

$$Worst_{MTTD} = T_{f\_scan} * N_{frame}$$

With T_{f_scan} the time to readback one frame from SRAM and to verify its hamming syndrome, N_{frame} is the number of frames to scan.

B. Checkpoint/Recovery Strategy

Generally, checkpoint can be either transparent or intrusive. Depending on the executive platform: the first case assumes that the checkpoint of a module can be triggered while it is performing its computation; while the second case assumes that it must be suspended to construct checkpoint. In transparent checkpoint, a mechanism to make a copy of all the information of the context must be made available within the executive platform. The checkpoint method used in our reliability controller is able to achieve this type of transparent checkpoint, it is a direct consequence of these facts: first, the context of a module running on SRAM-based FPGAs is located on flip-flops (and Block RAM in some cases but for the moment we treat only flip-flops); and second, the ability to copy the content of each flip-flip from the operative layer to the configuration layer at the same time. Also, our reliability controller can perform checkpoint in two ways: periodic or random, which depends on the: nature of modules and it's context, variation of fault rate and duration of checkpoint. By using partial dynamic reconfiguration, we provide a way to make checkpoint and recovery of individual ERR. This is possible by masking/unmasking online the frames involved in the operation of capture/restore of flip-flops. The mask/unmask operation consists on writing a special frame per column of resources that contains configuration bits for partial reconfiguration. One of these bits can mask the order of capture/restore of flip-flops [20].

The checkpoint performed by the reliability controller is divided into these basic actions:

- Unmask frames involved on the checkpoint.
- Order the capture of the flip-flops to the SRAM.
- Readback the unmasked frame out of SRAM.
- Mask again the frames.

In literature, a checkpoint is characterized by two metrics [23], a *checkpoint latency*: that represents the quantity of time needed to save a checkpoint on a safe storage not subject to faults, denoted as L; and a *checkpoint overhead*: which represents the introduced growth in the execution time of the module induced by the checkpoint operation, denote as C. In our method the time for capturing the state of flip-flops to the configuration layer is zero (C=0). Let T_{f_read} the time to readback one frame from SRAM, the latency is given by:

$$L = T_{f\_read} * N_{frame}$$

We notice that in our case C is zero because the checkpoint is transparent and is performed instantaneously during the module execution.

Upon a detection of faulty frame, a recovery is needed to be triggered. It induces a loss of computation performed from the last checkpoint until that moment of detection, this time is denoted T_{lost}. Also an overhead is needed to write back the checkpoint to the SRAM, this time is calculated by:

$$T_{rollback} = T_{f\_write} * N_{frame}$$

where T_{f_write} is the time to write a frame to the SRAM. As checkpoint, the operation of recovery is divided onto these actions:

- Write the checkpoint to frames on SRAM.
- Unmask frames to roll back.
- Order the restore of the flip-flops from the SRAM.
- Mask again the frames.

We denote the complete overhead of a recovery from an error as R, which is the sum of lost time computation and roll back overhead.

$$R = T_{rollback} + T_{lost}$$

In figure 4, a representation of a timeline of a module and the reliability controller execution are given to illustrate the previous definitions related to detection, checkpoint and recovery, a worst scenario is presented where the recovery occurs just before a planned checkpoint.

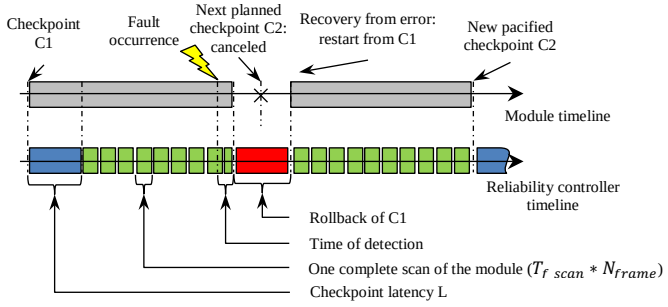

Figure 4. Outline of execution scenario.

It is clear that a bit-flip can occur between the moment when capture action is triggered and the end of read back of all frames of the ERR, to prevent this from happening an additional comparison is performed between the frames of the initial bitstream and the read frames from SRAM (we perform a bit mask on flip-flops before making the comparison because flip-flops have necessarily changed).

These two presented strategies of detection and checkpoint-recovery operate following the flowchart presented in figure 5. First, the initialization of the BER strategy is done and a first checkpoint is created. After, a detection phase is performed continually by readback and hamming verification. This combination leads to minimize the detection time of a fault, which is always less or equal to $Worst_{MTTD}$.

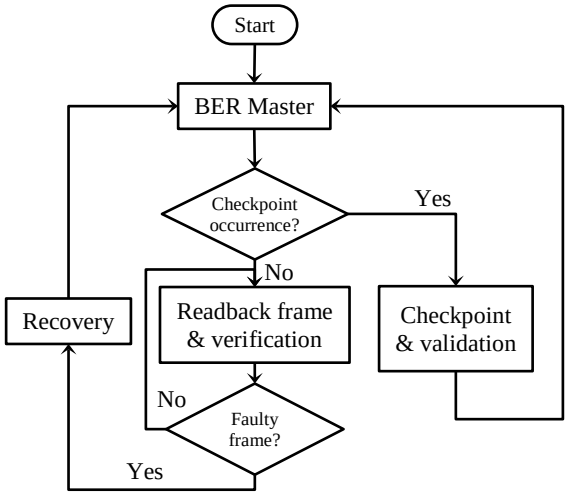

Figure 5. Flowchart of the reliability controller.

C. *Reliability Controller Implementation*

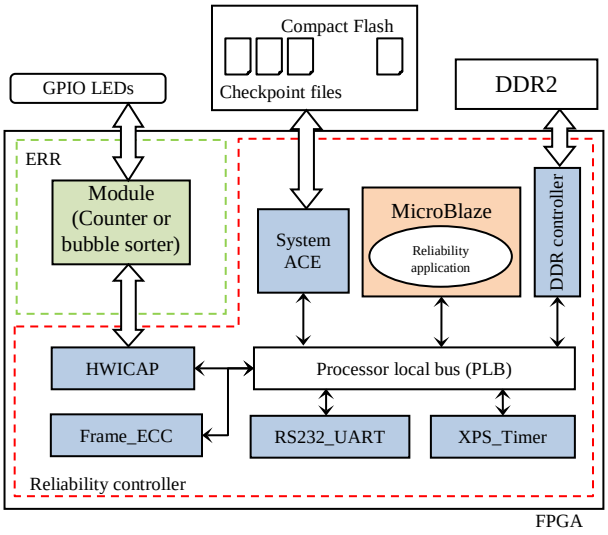

Figure 6. Checkpoint/Recovery Platform.

To rapidly validate our approach we implement it using the soft-core processor MicroBlaze with fault tolerant features enabled. This later is running a software version of the flowchart presented in figure 5. The controller contains a HwIcap to communicate with the Configuration Control Logic (CCL) in order to read/write frames from/to the configuration layer. The primitive Frame ECC [20] is used to check the hamming syndrome for each frame in the detection phase; the output of this primitive is updated each time a readback of one frame is performed. A SystemACE is also used to provide access to external memory (safe storage) where context of modules can be saved for later use. A PLB bus is added to provide a way for the processor to communicate with the other parts of the controller. To collect information about the execution an RS232 UART is added and connected to an output terminal. Time measurement is performed with an XPS_Timer and is clocked like the MicroBlaze at 100 MHz.

IV. EXPERIMENTAL RESULTS

Our experiments have been tested on ML506 board which incorporates Virtex-5 FPGA [20], the tools used are Xilinx ISE Design Suite (13.1). We applied our method on two hardware modules: a binary counter and an implementation of the bubble sort algorithm. We performed a random fault injection from the MicroBlaze to perform the simulation of faults occurrence into the SRAM region of the ERR. Table I presents the resources utilization of the reliability controller and the tested hardware module.

TABLE I
Resources utilization for reliability controller on virtex-5

Virtex-5 FPGA	Look-up tables (LUTs)	Flip-flops (FFs)
(xc5vsx50tff1136-1)	32640	32640
System utilization	6669 (21%)	7772 (24%)
Binary Counter	73	36
Bubble sorter	444	198

This software version of the proposed method uses less than 25% of the resources and leaves the rest of them for the implementation of the enhanced reliable region, which can be larger on the recent FPGAs like Virtex-6 and Virtex-7. The timing measurements of the method are presented on Table II.

TABLE II
Timing measurements of operation of reliability controller (Millisecond)

Operations	Details of operation	For one frame	Binary Counter: 36 frames	Bubble Sorter: 144 frames
Detection strategy	SRAM Read Frame	0.82	29.73	119.18
	Syndrome verification	0.08	-	-
Checkpoint strategy	Flip-flop Capture*	0.12	0.12	0.12
	SRAM Read Frame	0.82	29.73	119.18
	SysACE write	14.24	510.24	2050.7
Recovery strategy	SysACE read	11.97	431.01	1723.6
	SRAM Write Frame	1.01	36.3	144.14
	Flip-Flop Restore*	0.09	-	-

* The time here reflect the time MicroBlaze takes to trigger the order and not the time of capture/restore.

We point out that the frames read/write operations are the ones taking a great portion of the overall execution time of each strategy (the SysACE read/write can be performed by a dedicated controller) and that the speed of the HwIcap is a bottle neck for this method. For our system both configuration and readback can be minimized by using dedicated hardware controller such as UPaRC [24] or FaRM [25], which performs partial dynamic reconfiguration at high speed and takes into account the power consumption factor. The footprint of a checkpoint can be optimized by just saving the binary differences between the initial bitsream and the read back one. Such optimization will cost a computation overhead which can be accepted when it is smaller than the total overhead of writing to the safe storage.

V. CONCLUSION AND FUTURE WORK

This paper presented a new fault tolerant method based on partial dynamic reconfiguration. The method uses error detection and correction codes (EDAC) to localize faults and it uses checkpoint/recovery to correct them. The actual implementation of the method confirms advantages of such an approach: non-intrusive and efficient fault correction. Such a method can be applied with soft real-time applications to satisfy fault tolerance.

We are actually working on testing this method with different types of modules: combinatorial, sequential and mixed hardware IP. We are also planning to provide an automatic method to evaluate the best moment to trigger the checkpoint of modules.

REFERENCES

[1] Xilinx, "Partial Reconfiguration User Guide UG702 (v12.1)," May 3, 2010.

[2] A. Corporation, "Increasing Design Functionality with Partial and Dynamic Reconfiguration in 28-nm FPGAs," White paper, 2010.

[3] E. J. McDonald, "Runtime FPGA Partial Reconfiguration," in *Aerospace Conference, 2008 IEEE*, 2008, pp. 1-7.

[4] Y. Iskander, S. Craven, A. Chandrasekharan, S. Rajagopalan, G. Subbarayan, T. Frangieh, and C. Patterson, "Using partial reconfiguration and high-level models to accelerate FPGA design validation," in *Field-Programmable Technology (FPT), 2010 International Conference on*, 2010, pp. 341-344.

[5] H. Chuan, K. Benkrid, X. Iturbe, A. Ebrahim, and T. Arslan, "Efficient On-Chip Task Scheduler and Allocator for Reconfigurable Operating Systems," *Embedded Systems Letters, IEEE*, vol. 3, pp. 85-88, 2011.

[6] L. Rockett, D. Patel, S. Danziger, B. Cronquist, and J. J. Wang, "Radiation Hardened FPGA Technology for Space Applications," in *Aerospace Conference, 2007 IEEE*, 2007, pp. 1-7.

[7] C. C. Xapp, "Triple Modular Redundancy Design Techniques for Virtex FPGAs," July, 2006.

[8] J. Johnson, W. Howes, M. Wirthlin, D. L. McMurtrey, M. Caffrey, P. Graham, and K. Morgan, "Using Duplication with Compare for On-line Error Detection in FPGA-based Designs," in *Aerospace Conference, 2008 IEEE*, 2008, pp. 1-11.

[9] J. A. Cheatham, J. M. Emmert, and S. Baumgart, "A survey of fault tolerant methodologies for FPGAs," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 11, pp. 501-533, 2006.

[10] S. Yousuf, A. Jacobs, and A. Gordon-Ross, "Partially reconfigurable system-on-chips for adaptive fault tolerance," in *Field-Programmable Technology (FPT), 2011 International Conference on*, 2011, pp. 1-8.

[11] M. A. Sullivan, H. H. Loomis, and A. A. Ross, "Employment of Reduced Precision Redundancy for Fault Tolerant FPGA Applications," *Field-Programmable Custom Computing Machines, Annual IEEE Symposium on*, vol. 0, pp. 283-286, 2009.

[12] M. Lanuzza, P. Zicari, F. Frustaci, S. Perri, and P. Corsonello, "Exploiting Self-Reconfiguration Capability to Improve SRAM-based FPGA Robustness in Space and Avionics Applications," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 4, pp. 1-22, 2010.

[13] J. Heiner, B. Sellers, M. Wirthlin, and J. Kalb, "FPGA partial reconfiguration via configuration scrubbing," in *Field Programmable Logic and Applications, 2009. FPL 2009. International Conference on*, 2009, pp. 99-104.

[14] D. K. Bhattacharryya and S. Nandi, "An efficient class of SEC-DED-AUED codes," in *Parallel Architectures, Algorithms, and Networks, 1997. (I-SPAN '97) Proceedings., Third International Symposium on*, 1997, pp. 410-416.

[15] B. Dutton and C. Stroud, "Built-in self-test of embedded seu detection cores in virtex-4 and virtex-5 fpgas," 2009, pp. 149-155.

[16] M. Berg, C. Poivey, D. Petrick, D. Espinosa, A. Lesea, K. A. LaBel, M. Friendlich, H. Kim, and A. Phan, "Effectiveness of Internal Versus External SEU Scrubbing Mitigation Strategies in a Xilinx FPGA: Design, Test, and Analysis," *Nuclear Science, IEEE Transactions on*, vol. 55, pp. 2259-2266, 2008.

[17] D. Koch, C. Haubelt, and J. Teich, "Efficient hardware checkpointing: concepts, overhead analysis, and implementation," presented at the Proceedings of the 2007 ACM/SIGDA 15th international symposium on Field programmable gate arrays, Monterey, California, USA, 2007.

[18] A. G. Schmidt, H. Bin, R. Sass, and M. French, "Checkpoint/Restart and Beyond: Resilient High Performance Computing with FPGAs," in *Field-Programmable Custom Computing Machines (FCCM), 2011 IEEE 19th Annual International Symposium on*, 2011, pp. 162-169.

[19] C. Bolchini and C. Sandionigi, "Fault Classification for SRAM-Based FPGAs in the Space Environment for Fault Mitigation," *Embedded Systems Letters, IEEE*, vol. 2, pp. 107-110, 2010.

[20] Xilinx. *Virtex-5 FPGA Configuration User Guide UG191 (v3.10)*. Available: www.xilinx.com/support/documentation/user_guides/ug191.pdf

[21] Z. Wang, Z. Yao, H. Guo, and M. Lv, "Bitstream decoding and SEU-induced failure analysis in SRAM-based FPGAs," *SCIENCE CHINA Information Sciences*, pp. 1-12, 2011.

[22] J.-B. Note and É. Rannaud, "From the bitstream to the netlist," presented at the Proceedings of the 16th international ACM/SIGDA symposium on Field programmable gate arrays, Monterey, California, USA, 2008.

[23] N. H. Vaidya, "Impact of checkpoint latency on overhead ratio of a checkpointing scheme," *Computers, IEEE Transactions on*, vol. 46, pp. 942-947, 1997.

[24] R. Bonamy, P. Hung-Manh, S. Pillement, and D. Chillet, "UPaRC - Ultra-fast power-aware reconfiguration controller," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2012*, 2012, pp. 1373-1378.

[25] F. Duhem, F. Muller, and P. Lorenzini, "Reconfiguration time overhead on field programmable gate arrays: reduction and cost model," *Computers & Digital Techniques, IET*, vol. 6, pp. 105-113, 2012.