Design Techniques for Xilinx Virtex FPGA Configuration Memory Scrubbers

I. Herrera-Alzu and M. López-Vallejo

Abstract—**SRAM-based FPGAs are in-field reconfigurable an unlimited number of times. This characteristic, together with their high performance and high logic density, proves to be very convenient for a number of ground and space level applications. One drawback of this technology is that it is susceptible to ionizing radiation, and this sensitivity increases with technology scaling. This is a first order concern for applications in harsh radiation environments, and starts to be a concern for high reliability ground applications. Several techniques exist for coping with radiation effects at user application. In order to be effective they need to be complemented with configuration memory scrubbing, which allows error mitigation and prevents failures due to error accumulation. Depending on the radiation environment and on the system dependability requirements, the configuration *scrubber* design can become more or less complex. This paper classifies and presents current and novel design methodologies and architectures for SRAM-based FPGAs, and in particular for Xilinx Virtex-4QV/5QV, configuration memory *scrubbers*.**

Index Terms—**Field Programmable Gate Array (FPGA), reconfiguration, scrubbing, single event upset, Xilinx.**

I. Introduction

When compared to other FPGA technologies, SRAM-based FPGAs allow high performance, high logic density and low Non-Recurring Engineering (NRE) costs. At the same time, the FPGA can be statically reconfigured, after the initial power-on configuration, practically an unlimited number of times. Moreover, some FPGAs can be partially reconfigured during run-time without interrupting the application. Dynamic partial reconfigurability can be exploited in different ways, as will be seen in this paper. These are valuable features for a number of applications, for example those requiring reconfigurable processing or remote configuration upgrades. In particular, interest of re-programmable FPGAs for space applications has been shown since more than one decade ago [1]–[4]. Because configuration memory is volatile, an external device needs to take care of power-on and subsequent reconfigurations. In some cases this device can be simply an external non-volatile memory that provides configuration data for FPGA to self-reconfigure. In others, a higher level of intelligence is required.

On the other hand, when designing for applications where ionizing radiation is relevant (e.g., space, medical or nuclear power plants) the device radiation tolerance is a first order concern. SRAM-based FPGAs are known to be sensitive to radiation, so in order to benefit from the aforementioned features also in harsh radiation environments, their radiation tolerance needs to be improved by design. The FPGA architecture and the radiation effects on it need to be well understood for implementing effective error mitigation techniques. In particular, mitigation techniques for the FPGA configuration layer are addressed in this paper.

Scrubbing is an effective error mitigation technique for configuration memory in SRAM-based FPGAs. It consists on a post-configuration write of the configuration memory to restore its initial state. This can be done in SRAM-based FPGAs without interrupting the system operation, and aims at mitigating errors before their accumulation induces a system failure. Circuitry performing such task is informally known as *scrubber*. Depending on the system-level constraints and on the radiation environment for the application, different *scrubber* implementation options exist involving more or less overhead in terms of complexity, area and power consumption.

In spite of the complexity added to implement error mitigation, the benefits of SRAM-based FPGAs are considered to be dominant in many applications. Alternatives to this technology for harsh radiation environments are basically anti-fuse (ROM-based) and Flash-based [4]. Anti-fuse technology is inherently radiation-tolerant but, while suitable for many applications, it does not support in-field reconfiguration. Regarding flash technology, its TID limitations and potential charge leakage is undergoing scrutiny by the industry. Besides, for the moment it does not support dynamic partial reconfiguration. Finally, among SRAM alternatives, Xilinx Virtex-4QV/5QV are found to have the highest logic density, performance and radiation-tolerance altogether. Therefore, the present work will focus on the latter FPGAs. An in-depth comparison of FPGA technologies for harsh radiation environments is out of the scope of this paper and can be found in the literature [5], [6].

The main contribution of the present work has been to identify, classify and compare the main trends in configuration scrubbing techniques, methodologies and architectures for SRAM-based FPGAs (and in particular for Xilinx Virtex series), to provide the designer with criteria for trading-off the different options, and to provide system-level considerations when reliability and/or power consumption are design drivers. The paper is organized as follows. Section II recalls the Xilinx Virtex FPGA architecture. Section III summarizes the Radiation Effects on the Xilinx Virtex FPGA. Section IV presents the Configuration Memory Scrubbing Overview. Section V presents the Configuration Memory Scrubbing Basics. Section VI presents usual and novel Scrubbing Methodologies. Section VII presents the elementary Scrubber Design

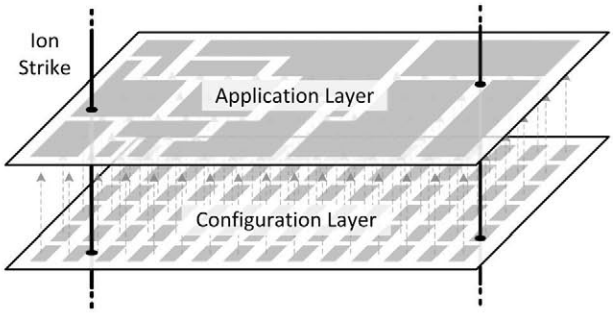

Fig. 1. Xilinx Virtex conceptual layers: Application Layer (user logic and memory) and Configuration Layer (logic and routing resources configuration).

Architectures. Section VIII discusses main System-Level Considerations, in particular Reliability, Availability and Power Consumption. Finally, Section IX draws some Conclusions.

II. Xilinx Virtex FPGA Architecture

Xilinx Virtex FPGAs can be conceptually split in two layers, namely Application Layer (from now on, A-Layer) and Configuration Layer (from now on, C-Layer). A-Layer includes the logic and memory elements managed by user's application and C-Layer includes the logic and memory elements that allow configuring the logic and routing resources in the A-Layer. These layers are represented in Fig. 1.

A. Application Layer

The A-Layer comprises the logic, memory and input/output resources for user application. Most of the layer implements Configurable Logic Blocks (CLBs), which can be configured to implement any user sequential or combinational circuit. Each CLB in turn consists of several *slices*, and each *slice* contains Look-Up Tables (LUTs), registers and carry logic. LUTs can be eventually used as distributed RAM resources (LUTRAMs) or shift registers (SRL16/32). Some static control signals have pre-assigned logic levels via weak keeper pull-ups and pull-downs known as *half-latches*, wich can be forced to a hard logic level via internal routing.

The other fundamental components are the Input/Output Blocks (IOBs). Each IOB can be configured for a wide variety of interface standards and voltage levels. This layer also implements a number of application resources depending on the specific FPGA, being the most relevant the following: dedicated dual-port SelectRAM blocks (BRAMs), dedicated multipliers (physically located next to BRAMs) and DSP blocks, Digital Clock Managers (DCMs), embedded PowerPC processors (PPC), Multi-Gigabit serial Transceivers (MGT) and 10/100/1000 Ethernet MAC (EMAC). Finally, a large General Routing Matrix (GRM) connects all the elementary blocks.

Two main Xilinx Virtex-4 and -5 radiation-tolerant families have been released so far, namely Virtex-4QV and -5QV (or shortly, XQR4V and XQR5V). They are manufactured on a 90-nm and 65-nm CMOS process, respectively, on thin epitaxial silicon wafers. Besides the manufacturing process, the XQR4V parts are functionally identical to their commercial counterparts. However the XQR5V design has been optimized for a better radiation tolerance, or as the manufacturer claims, it has been Radiation Hardened By Design (RHBD). The resources available for user application are summarized in Table I.

TABLE I
Xilinx Virtex XQR4V and XQR5V Resources

Resource Type	XQR4V				XQR5V
	SX55	FX60	FX140	LX200	FX130
Config Mem (Mbit)	15.4	14.5	34.5	43.0	49.2
BRAM (Mbit)	5.9	4.3	10.2	6.2	10.7
Logic (Kslices)[a]	24.6	25.3	63.2	89.0	20.4
DSP (Blocks)[b]	512	128	192	96	320
PPC (Blocks)	-	2	2	-	-
DCM (Blocks)	12	12	20	12	12
MGT (Blocks)	-	-	-	-	18
EMAC (Blocks)	-	-	-	-	6
IOB (Blocks)	640	576	896	960	836

[a]XQR4V slices have 2 4-input LUTs and 2 registers. XQR5V slices have 4 6-input LUTs and 4 registers.

[b]XQR4V DSP blocks have 18×18 multipliers. XQR5V DSP blocks have 25×18 multipliers.

B. Configuration Layer

The C-Layer comprises the configuration memory and associated access ports and control logic. Virtex-4 FPGAs implement the following access ports for reading and writing from/to configuration memory: JTAG (Joint Test Action Group), master/slave serial and master/slave SelectMAP (Selectable Microprocessor Access Port), the latter available in 8 and 32-bit bus width. In addition to them, Virtex-5 FPGAs implement master SPI (Serial Peripheral Interface), master BPI (Byte Peripheral Interface) and slave SelectMAP in 8/16/32-bit bus width. According to the manufacturer [7], SelectMAP provides the most efficient device access for scrubbing, and the 8-bit bus width is typically used for *scrubber* implementations as it does not require data word alignment. In Virtex-5, a bus width auto-detection feature has been added, thus simplifying the implementation of *scrubbers* for 16/32-bit bus widths.

The configuration port is internally connected to a built-in circuitry that decodes the configuration data packets, providing read/write access to the configuration control registers and to the configuration memory. In addition to the configuration ports, the configuration controller can be reached from within the A-Layer via the Internal Configuration Access Port (ICAP). This allows the implementation of *scrubbers* internal to the FPGA.

Major architectural changes were implemented in Virtex-4 C-Layer with respect to its predecessors: *frame* size was reduced and kept uniform across the device, *frames* were distributed into rows allowing a 2D mapping of configuration memory, built-in masking of dynamic memory elements was implemented allowing transparent readback, and Error Correction Code (ECC) was embedded into the *frame* structure. Further architectural improvements were implemented in Virtex-5, specifically built-in blocks supporting Cyclic Redundancy Check (CRC) computations. ECC and CRC will be described in Section V.A.

III. Radiation Effects on Xilinx Virtex FPGAs

SRAM-based FPGAs are sensitive to ionizing radiation effects, which induce both a long term cumulative degradation (Total Ionization Dose, or TID, and displacement), as well as

instantaneous damage (Single Event Effects, or SEE). SEE can in turn be classified into *soft errors*, which affect data integrity, and *hard errors*, which damage silicon structures. *Soft errors* are reversible, like Single Event Upset (SEU), Multiple Bit Upset (MBU) or Single Event Transient (SET), but *hard errors* can be destructive, like Single Event Latch-up (SEL). In particular, commercial Xilinx Virtex FPGAs are also sensitive to some SEE types. For harsh radiation environment applications, the XQR4V/5V families offer improved protection against SEL and higher TID levels. For XQR4V, C-Layer SEUs are still a concern as they are a major contributor to system failure rate. A-Layer SEUs (but not MBUs) can be mitigated with different Triple Modular Redundancy (TMR) strategies. For XQR5V, the manufacturer claims that C-Layer SEUs are 1000 × lower than in XQR4V [8]. In addition, architectural improvements in this device, such as built-in Error Detection and Correction (EDAC) for BRAM and SET filters for flip-flops, allow reducing A-Layer upsets significantly.

When high-energy particles strike the FPGA, *soft errors* can appear either in A-Layer or in the underlying C-Layer. When in A-Layer, *soft error* effects can be transient or persistent, depending on whether the affected logic has memory (sequential logic, flip-flops, BRAMs, *half-latches*) or is memoryless (combinational logic). These *soft errors* can lead to Single Event Functional Interrupts (SEFIs) affecting part of the device or the whole of it. When in C-Layer, *soft errors* can affect configuration logic or routing resources, and their effects are persistent until a partial or global reconfiguration restores the initially configured value. In the case of *soft errors* affecting the configuration controller or other built-in functions, SEFIs may also occur and can only be recovered by FPGA re-initialization.

The functional effects of *soft errors* in logic resources have been extensively researched [9], [10]. XQR4V has been experimentally characterized for both static and dynamic *soft errors* [11], [12]. For XQR5V, some SEU/SEFI figures can also be found in [8], [13] and [14]. *Soft errors* in routing resources, even if not used by the A-Layer, may also have functional effect as they may create shorts or bridges between logic resources. Indeed, *soft errors* in routing resources may induce single or multiple errors in the A-Layer [15]. The situation is dramatic considering that more than 60% of the configuration bits in modern Virtex FPGAs are linked to routing resources, although 10 to 20% of them are typically used. On the other hand, configuration SRAM density keeps on growing in every new FPGA generation, thus increasing the MBU cross-section [16], [17].

Radiation effects must be assessed for every application, as they depend on the particular radiation environment, on the specific FPGA, and on the specific design implemented on the FPGA. The FPGA SEU rate can be estimated using tools like CREME96 [18], which provides orbit-averaged static SEU rates from FPGA's static cross-section data. For XQR4V/5V FPGAs, the cross-section data can be obtained from the test reports produced by members of the Xilinx Radiation Test Consortium (XRTC) [19]. Fault injection methods [20]–[22], can be used to provide a qualitative view on how well a design is mitigated [23], but the dynamic SEE characterization on the specific design implementation can only be obtained experimentally by accelerator testing.

IV. Configuration Memory Scrubbing Overview

During the past decade, the research community has proposed different solutions for coping with Virtex FPGA configuration memory SEUs. This work was also fostered by the research on dynamic reconfiguration techniques and fault injection systems, and it is continuously supported by industry, government and academia via the XRTC.

Complementary *soft error* tolerance techniques have been incorporated to enable the use of Virtex FPGAs in harsh radiation environments. EDAC techniques, such as TMR and ECC codes, are typically used in user logic and memories for preserving design functionality and data integrity in the event of SEUs. On the other hand, the dynamic reconfigurability of the configuration memory can be used, not only for reconfiguring or upgrading the application, but also for mitigating *soft-errors* before they accumulate potentially defeating EDAC techniques implemented in A-Layer. This technique, widely used in SRAM memories, is known as memory scrubbing. Both TMR and configuration memory scrubbing, when used jointly, provide the highest SEU mitigation capability [24]. However, the configuration memory scrubbing has some limitations in what concerns error detection and mitigation, as will be explained in Section V.D.

Scrubbing is basically a post-configuration memory refresh. Typically it is performed in the background without disrupting the A-Layer. For this purpose the FPGA initialization commands in the configuration *bitstream* are omitted, and only those required to initialize the scrubbing are kept. In other cases, the full *bitstream* is injected in the C-Layer and the application is momentarily suspended then reinitialized. On the other hand, readback is a post-configuration memory read. Also performed in the background, it does not disrupt the application but it may corrupt the data content of some memory elements in early Virtex families, as will be explained in Section V.D.

Scrubbing and readback are handled by an error detection and correction circuitry, sometimes referred as Configuration Manager and informally known as *scrubber*. Depending on system dependability requirements, *scrubbers* can implement anything from simple preventive mitigation to complex adaptive reconfigurable strategies. Some basic scrubbing techniques for Virtex-4 family are introduced in [7]. These techniques are still valid for Virtex-5 family, although they need to be adapted to the changes introduced in the configuration logic. Main guidelines on radiation effect mitigation for Xilinx FPGAs up to Virtex-4 are found in [10], which focuses mainly on TMR implementation. A follow-up work was found in [23], where both SEFI and SEU detection, mitigation and mitigation verification are covered in great detail. Regarding Virtex-5, new strategies for SEU handling are proposed by Xilinx in [25]. The main difference with respect to methods proposed for Virtex-4 is the use of built-in circuitry for error detection and correction, as will be explained in Section V.A.

V. Configuration Memory Scrubbing Basics

Upon SEEs in C-Layer, a generic *scrubber* has two main duties: to detect the SEEs and to mitigate them before they accumulate and/or disrupt the application. The detection phase is optional and requires additional complexity, but it generally

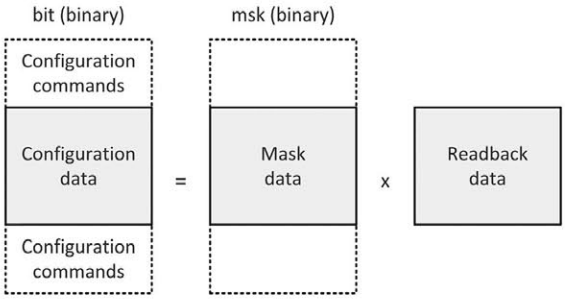

Fig. 2. Configuration *bitstream* file (bit), mask file (msk) and readback data relationship: masked readback data matches configuration data if no error is present.

allows a more robust mitigation. However in the highest configuration memory size devices, the complexity of the detection scheme may not be affordable. The application disruption is sometimes unavoidable, and the system needs to implement functional monitoring (e.g., external watchdog timer circuit) to detect it and recover the system.

A. SEU/MBU Detection

Configuration memory SEU/MBUs can be detected by reading back the FPGA configuration data via the configuration port, and comparing it with a golden reference stored in a non-volatile memory (from now on, *golden memory*). Xilinx development tools generate two files for each configuration *bitstream* file: a readback data file and a mask data file. The mask file is used to mask the readback bits that may change during operation. Although it is possible to use both the readback data and mask data files, the storage of the two files (in addition to the *bitstream* file) needs to be considered at system level, and may not be affordable for the most complex FPGA devices given the size of the files. A less stringent solution requires only the mask data file to be stored in addition to the *bitstream* file. Each masked readback word must be compared with the corresponding golden reference in the *bitstream*, as shown in Fig. 2. In case of mismatch, an error detection flag must be raised to trigger the corresponding mitigation actions. For XQR5V, and according to latest findings reported by the manufacturer in [8], the mask data file (as generated by automated tools) needs to be modified in order to mask non-configuration bits (in particular, capture bits) that are also part of the readback stream.

The word-by-word masking and comparison is a comprehensive but rather slow technique and requires accessing the full configuration *golden memory* for comparison. A faster technique consists on performing a CRC check on the readback data, which needs reference CRC codes (much less information than the configuration data) to be previously generated and stored in internal or external memory. Using for example 32-bit CRC codes, the probability of not detecting upsets is as low as 2^{-32}. CRC can be checked for each *frame* data, or for the full device data [26].

For the particular systems implementing device-level TMR (in addition to TMR within each FPGA), a 3-way *scrubber* can detect SEU/MBU in each FPGA by comparing readback data from each of the independent channels. This allows a fast detection as a fault-free *frame* is available in a temporary buffer instead of in external *golden memory*. The *scrubber* solution presented in [27] targets device-level TMR systems and propose this kind of solution combined with a self-repair approach.

A new feature in Virtex-4/5 with respect to its predecessors is that every configuration *frame* contains a 12-bit built-in ECC code, which uses Single Error Correction Double Error Detection (SECDED), Hamming code parity values. In case a *frame* bit flips due to SEU, the ECC check will allow detection and identification of a single bit in error within the *frame*, which can be used for SEU mitigation. Double error or more can be detected but the ECC check does not identify the bits in error. ECC bit flips do not affect the A-Layer as they do not configure any resource, but may result in bogus SEU detection and the corresponding mitigation may corrupt the *frame*. Virtex-5 also provides a built-in readback CRC circuit for overall memory error detection. This circuit, clocked internally or externally, continuously scans the configuration memory and computes a 32-bit CRC value. The CRC value computed right after device configuration is used as the fault-free reference for comparing the values obtained in subsequent scans. In case of mismatch, a device pin is driven low to report the configuration error to an external supervisor.

B. SEU/MBU Mitigation

SEU/MBU in configuration memory are mitigated by means of static or dynamic reconfiguration. The former implies stopping the application and performing a full configuration, whereas the latter is to be performed in the background, for the full device or part of it, while the application is running. The minimum reconfigurable element is a *frame*. If the detection method provides the address of the *frame* in error, the reconfiguration can then target that particular *frame*. If the faulty *frame* address is not available, a full device reconfiguration is needed. The former may be preferred, as it generally reduces the risk of corrupting the configuration memory. Moreover, from a reliability point of view, the time-to-repair is shorter, as will be shown in Section VIII.A.

C. SEFI Detection and Recovery

Different SEFI modes were found in XQR4V FPGAs during radiation testing [11], and a few of them were also found in XQR5V [8]. Some can be detected by using an external supervisor, like a watchdog-timer, although others require a more complex diagnostic hardware, optionally included in the external *scrubber*. The exact SEFI mode may be difficult to identify due to the erratic behavior of the device. Some of them can be detected by periodic polling of critical configuration logic registers, like Frame Address Register (FAR), Status (STAT) and Control (CTL) registers, and comparison with golden reference values. For example, a continuous increase of the FAR indicates that the built-in configuration sequencer is out of control, which in turn may prevent any attempt for the *scrubber* to read or write configuration memory. To detect this type of SEFI, a write-read-check test is recommended before starting any readback or scrub sequence [7].

Other SEFIs may be detected by monitoring configuration port pin voltages. For example, a transition to logic low in the DONE pin indicates that the device configuration is lost (or shutdown sequence is on-going), due to the trigger of the Power-On-

Reset (POR). In most cases, a full reconfiguration is needed to recover from SEFI. Power cycling can be optionally used, although in principle it is not needed [23].

D. Scrubbing Limitations

Virtex FPGA SEU/MBU detection and mitigation via readback and scrubbing has some inherent limitations:

- First, errors in hidden resources may propagate across the A-Layer without being detected by configuration memory readback. For example, *half-latches* cannot be read back nor scrubbed. Error mitigation techniques at A-Layer must be able to cope with them.
- Second, some logic resources cannot be readback because they can get corrupted. For example, SRL16s and LUT-RAMs readback and scrubbing is not allowed in early Virtex families. This was fixed in Virtex-4/5 by setting a specific configuration register bit (GLUTMASK).
- Third, some built-in circuitry in Virtex FPGAs cannot be scrubbed because they are not accessible. For example the configuration controller and the DCM.
- Fourth, some type of errors are persistent and cannot be mitigated only by reconfiguration: a system reset is also required [28]. Fortunately the persistent error cross-section is orders of magnitude lower than the one for non-persistent errors (which do not require system reset).
- Finally, SEU effects in the A-Layer may not be mitigated when they affect more than one logic path of a TMR system within the same scrub period.

VI. Scrubbing Methodologies

Basic scrubbing techniques described in Section V can be used in different ways, leading to different methodologies. Choosing the right methodology for an application typically implies trading-off reliability, complexity, flexibility, power consumption and cost. Several trade-offs are presented in this section, providing the designer with system level considerations to be taken into account.

A. Preventive vs. Corrective Scrubbing

The simplest scrubbing methodology consists on reconfiguring cyclically the FPGA, without performing any error detection whatsoever. This is referred as preventive or *blind* scrubbing [29], and such a *scrubber* is said to operate in open-loop. Thanks to the Virtex FPGA dynamic reconfiguration capability, the A-Layer operation does not need to be interrupted while being scrubbed. In some applications the scrubbing can be scheduled periodically, for example just before starting an operation period. In that case, a full configuration can be performed without caring about A-Layer interruption. With or without interrupting the operation, the *scrubber* would operate in write mode, so any unexpected error during the initial FAR configuration could potentially corrupt the configuration memory. Fortunately, the probability of such event is extremely low [30]. In order to minimize the impact of such event, *frame*-oriented blind scrubbing could be implemented instead of device-oriented.

Another methodology consists on reading back cyclically the configuration memory, and triggering the scrubbing only in case an error is detected. The readback and subsequent scrubbing can be either device or *frame*-oriented, as will be discussed in VI.B. Once the error scrubbing is completed, the readback process can be either resumed or restarted. This methodology is referred as corrective scrubbing, and such a *scrubber* is said to operate in closed-loop. Upon error detection, the A-Layer can be scrubbed with or without interrupting the operation. The *scrubber* would operate in read mode most of the time, in fact all of the time if no error is detected. It is only during scrubbing that configuration memory can potentially get corrupted, but the probability is considered much lower than in the preventive scrubbing case, and can also be limited to a single *frame* or a block of *frames*.

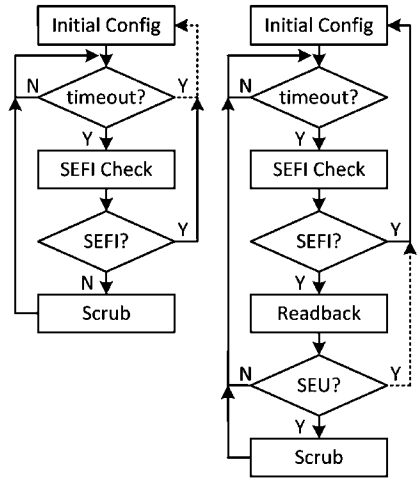

Fig. 3. Preventive vs. Corrective scrubbing flow. Preventive scrubbing (left) scrubs configuration memory periodically. Corrective scrubbing (right) reads back configuration memory periodically, and triggers scrubbing when error is detected. In both cases, the scrubbing can be done with (dashed line) or without (solid line) re-initiating the application.

A flow diagram for both preventive and corrective methodologies is shown in Fig. 3. It must be noted that the timeout checking is for the general case where the *scrubber* runs faster than the required scrub or readback rates. In some cases this checking is not needed, and the *scrubber* resources are full time devoted to readback and/or scrubbing.

B. Device vs. Frame-Oriented Detection and Mitigation

Configuration *frames* are typically read back or scrubbed sequentially by incrementing the FAR following a hardwired systematic sequence. This is automatically done by Virtex FPGA configuration controller when the number of read words exceeds the *frame* boundaries. Therefore, a simple open-loop *scrubber* can cyclically configure the scrub length for the total amount of configuration words, i.e., the total amount of configuration *frames* times the number of words per *frame*, plus a fixed overhead. This is considered a device-oriented scrubbing as far as error detection and mitigation covers the device as a whole. The scrub method is simple and has a minimum processing overhead, however it is exposed (with a very low probability) to the potential memory corruption, as highlighted in Section VI.A.

Likewise, a device-oriented readback can be cyclically configured. In case error is detected, scrubbing is triggered for the *frame* in error, for a block of *frames* or for all of them, then readback is resumed or restarted depending on the scrubbing strategy.

As opposed to the device-oriented approach, *frames* can be scrubbed or read back individually without using the built-in FAR auto-increment feature, which then needs to be implemented in the *scrubber* itself. The *scrubber* configures the readback length just for the amount of words in a *frame* (plus a fixed overhead). In this case, if an error is detected during readback, scrubbing is performed for just one *frame* before reading back the following *frame*. The readback and scrubbing functions are interleaved in what is considered a *frame*-oriented approach. This method has a higher overhead, but the mitigation time is minimized as well as corruption probability.

Intermediate approaches can be envisaged by dividing the *frame* space into blocks, and performing a block-oriented detection and mitigation. Then one must trade-off *frame* block length, readback overhead, detection and mitigation time and potential corruption probability and effects.

C. Fixed vs. Adaptive Rate

Scrub rate (for preventive mitigation) or readback rate (for corrective mitigation) was traditionally adjusted to a fixed rate of **10×** the maximum expected SEU rate along the mission, as recommended in [10]. This sometimes led to complex and power hungry *scrubber* implementations. Berg justified in [30] the need for a scrub rate of merely once every few days, which allows simpler *scrubbers*.

However there are two main reasons for considering a variable rate. The first is that the FPGA may not be in operational mode full time, being the rest of the time in idle, stand-by or even power-off mode. In stand-by mode, a higher SEFI rate may be tolerated (and mitigated by means of full reconfiguration), so readback or scrub rate could be reduced (or even stopped) in order to decrease system power consumption. Because SEUs may accumulate at both C-Layer and A-Layer during this mode, a full reconfiguration is deemed necessary before resuming operation.

The second is that radiation environment will vary over time during the mission. For example, when the spacecraft is crossing the South Atlantic Anomaly (SAA) region, an increase in particle flux is expected [31]. Another example is the peak radiation levels due to solar activity or Van Allen radiation belts. In all these cases, the readback rate could be temporarily increased in order to improve reliability and availability. Keeping the readback rate to the minimum required anytime during a mission helps optimizing the power consumption and heat dissipation, as will be explained in VIII.B. The need for adaptability was also identified in [32] and [33].

The related concept of Reconfigurable Fault Tolerance (RFT) was introduced by [33]. The objective of RFT is to enable a system to adapt to the optimal balance of performance and reliability during the mission. The *performability* metric, a combination of reliability and performance, shows improvements for applications using RFT architectures.

D. 1D vs. 2D Scrubbing

Device configuration *frames* are typically scanned sequentially along a 1D *frame* space. However, in a typical application, part of the configurable fabric is not used. SEUs in the configuration *frames* associated to unused logic may not be disruptive,

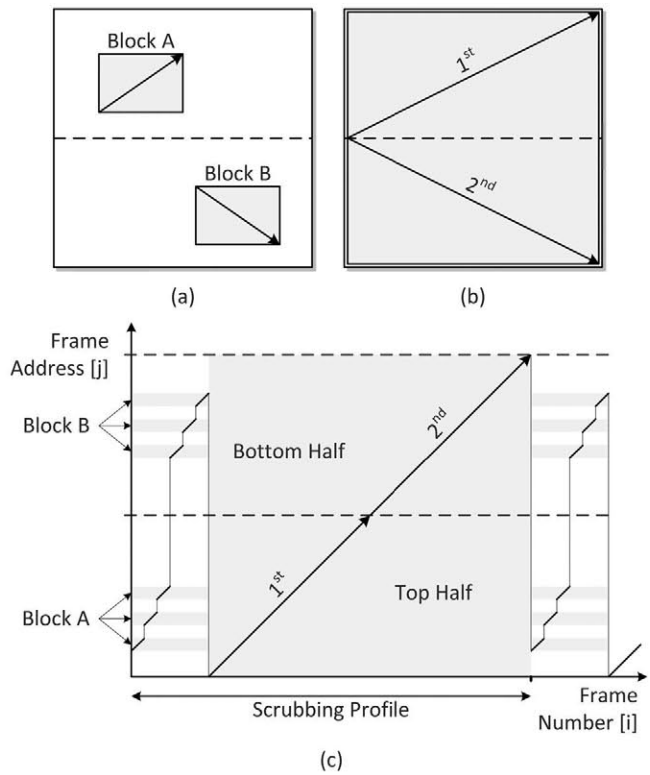

Fig. 4. Scrubbing Profile (SP) example. (a) Physical placement of Blocks A&B and readback or scrubbing direction, (b) Full FPGA readback or scrubbing direction, (c) SP sequence reading back or scrubbing Block A, then Block B, then full FPGA.

although there is still a possibility that they affect used routing resources. On the other hand, specific configuration *frames* may be more sensitive than others because they configure logic that is prone to propagate errors (e.g., voting logic in a TMR scheme). Such *frames* may be physically mapped into a 2D-block and read back (or scrubbed) at a higher rate.

A Scrubbing Profile (SP) is defined as the *frame* sequence followed during readback (or scrubbing) to cover the *frames* of interest, a certain number of times and in a certain order. A 2D-*scrubber* will not necessarily follow the *frame* address sequence pre-defined by the FPGA configuration logic, but the one defined by the SP. The SP is stored in the *scrubber*, it is repeated cyclically and it can be reprogrammed if needed. An SP example is shown in Fig. 4, where two FPGA blocks are read back (or scrubbed) at twice the rate than the rest.

1D or 2D scrubbing can be combined with any of the methodologies described in the previous three sections. For example, a *scrubber* may perform 2D readback (for corrective scrubbing) on a *frame*-oriented basis and with a fixed readback rate. And a different *scrubber* may perform 1D preventive scrubbing, on a device-oriented basis and with a variable readback rate. 1D/2D reconfiguration was also considered in [34] for block-based error mitigation.

VII. Scrubber Design Architectures

The scrubbing methodologies described in Section VI can be implemented in different *scrubber* architectures. Again, choosing the right architecture for an application typically implies trading-off reliability, complexity, flexibility, power

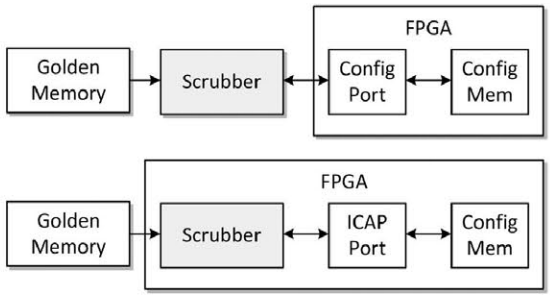

Fig. 5. External vs. Internal *scrubber*. External *scrubber* (up) accesses configuration memory via the configuration port. Internal *scrubber* (down) accesses configuration memory via Internal Configuration Access Port (ICAP).

consumption and cost. Some elementary architectures are presented in this section, with references to practical implementations found in previous works.

A. External vs. Internal Scrubber

Scrubbers are typically implemented in a radiation-hardened device external to the Virtex FPGA, such as an ASIC or an anti-fuse based FPGA. This ensures that *scrubber* logic itself is not affected by SEUs, or at least with a much lower probability than when embedded in the FPGA itself. *Scrubbers* internal to the Virtex FPGA may also be implemented by using the ICAP primitive, which provides internal access to the built-in configuration logic. However ICAP and internal logic are susceptible to radiation, so internal *scrubbers* are qualitatively less reliable than external ones, as was shown in [29]. Nevertheless, fault-tolerant implementations of ICAP controllers have also been investigated in [35]. An implementation combining internal *scrubber* and preventive scrubbing was found in [36], although it presents the limitations of both being internal and lacking error detection capability. Internal and external *scrubbers* are represented in Fig. 5.

Two *scrubber* implementations for Virtex-4 are compared in [29], the first one being internal to the FPGA itself and the second one implemented in an external FPGA. The internal implementation uses configuration memory readback in conjunction with SECDED. The external one simply reconfigures cyclically the configuration memory without any readback. Test results suggest a superior performance of the external implementation, however the limitations of such type of *scrubber* have been analyzed in Section VI.A.

An external *scrubber* in combination with TMR was successfully tested on a Virtex-II X-2V1000 device in [37]. The *scrubber* (referred as *configuration monitor*) was implemented on an auxiliary FPGA under the control of a host computer. This was continuously reading back, via SelectMAP or JTAG port, the configuration memory of the FPGA under test and comparing word-by-word with a mask file. Upon error detection, the auxiliary FPGA triggered a partial reconfiguration to mitigate the error. In addition, mechanisms were implemented to detect some types of SEFI and register corruption.

B. Hardware vs. Software-Based

Scrubbers can be implemented by hardware, either on an external or internal circuitry which defines a state machine for

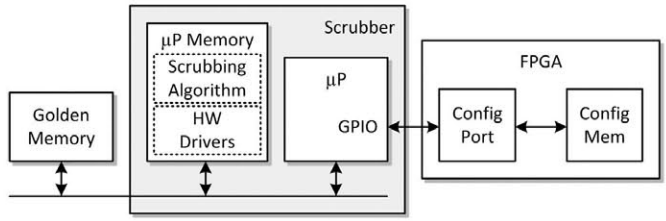

Fig. 6. Software-Based *scrubber*. *Scrubber* includes a microprocessor (μP) and its program memory. Program includes scrubbing algorithm and hardware drivers. μP accesses the FPGA via General Purpose Input/Outputs (GPIO).

readback and/or scrubbing. The main advantage of this architecture is that it can be very fast both in error detection and correction. However it lacks flexibility for implementing complex scrubbing strategies. Fig. 5 represents two typical hardware implementations based on radiation-hardened FPGA/ASIC (external) or embedded logic (internal).

Scrubbers can be also implemented on a radiation-hardened microprocessor system running a software algorithm. If the microprocessor has enough General Purpose Input/Output (GPIO) ports, it can directly drive the FPGA configuration port. Otherwise it needs an intermediate device to implement the hardware interface, as proposed in [38]. As shown in Fig. 6, the memory bus includes both the microprocessor memory (program and data) as well as the configuration *golden memory*. The software includes two main modules: the scrubbing algorithm (hardware independent) and the hardware drivers (specific for the microprocessor device and configuration port). The main advantage of this architecture with respect to a purely hardware one is the flexibility and the capability of implementing one or more complex algorithms. On the other hand, a software-based architecture is inherently slower. When implementing a corrective approach, the microprocessor needs many instruction cycles per *frame* to complete readback and error detection, which could be optimized if the microprocessor is devoted to the scrubbing function. If detection and mitigation is *frame*-based, execution time can be improved by pre-computing and storing in program memory the *frame* address sequence.

Hybrid HW/SW *scrubber* implementations can also be envisaged, for example with an FPGA embedding a microprocessor core running the scrubbing algorithm, and programmable logic for performing interface and data computation functions.

Most of the standalone *scrubbers* found in the literature are hardware-based, and a few software/firmware-based were found [39], [40]. This excludes systems for accelerator testing setup, which may involve a host PC running control software and interfacing the FPGA under test and associated logic.

C. One vs. N-Way

In a typical system, the *scrubber* has to manage a single FPGA configuration layer. This is a one-way architecture as opposed to others where the *scrubber* has to manage more than one. Systems implementing device-level Dual or Triple Modular Redundancy (DMR or TMR) are typical examples of the latter. In general, N-way architectures have to manage N configuration layers (i.e., channels), and those layers may belong to FPGAs implementing device-level redundancy or not.

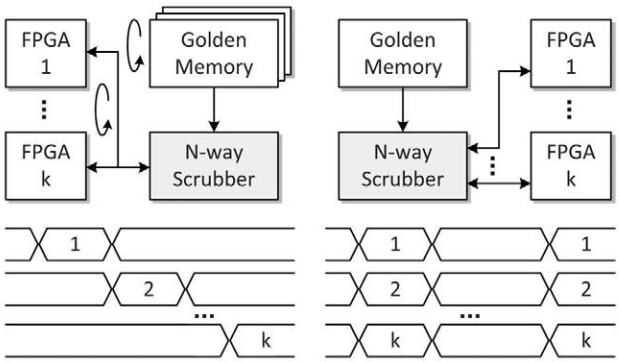

Fig. 7. N-way *scrubbers* managing k independent channels in time multiplex (*left*) and k independent channels concurrently (*right*).

In some cases, there are temporal constraints for which readback has to be concurrent across channels. This is the case for *scrubbers* that perform error detection by means of peer data comparison [27]. In that case, individual *scrubber* logic needs to be implemented for each channel. In the general case, when channels can be operated with functional and temporal independence from others, the same *scrubber* core logic and data bus lines can be shared. Only a few control lines need to be dedicated for each channel. Fig. 7 represents two generic N-way *scrubbers*, one of them managing k independent channels in time multiplex and the other one managing k redundant channels concurrently.

An external N-way *scrubber* was successfully tested on multiple Virtex V1000s implementing a space-based reconfigurable radio in [41]. The *scrubber* was implemented on a radiation-hardened anti-fuse FPGA, which was continuously scanning each of the nine Virtex V1000 via SelectMAP port. The anti-fuse FPGA computed the CRC, *frame* by *frame*, and compared it with a codebook of stored CRCs. Upon error detection, the anti-fuse FPGA interrupted the main microprocessor, triggered a partial reconfiguration to restore the *frame* in error, then reset the system.

VIII. System-Level Considerations

Scrubbing methodologies and architectures need to be analyzed together with three essential system-level design requirements: reliability, availability and power consumption. Of course, cost is also an important consideration but it is out of the scope of this paper. These requirements may impose additional constraints in the design of the configuration memory *scrubber*. In this section, a qualitative view is provided for each of them.

A. Reliability and Availability

Basic definitions for dependability metrics are given in [42]:

- Reliability or $R(t)$: the conditional probability that the system will perform correctly throughout the interval $[t_0, t]$, given that the system was performing correctly at time t_0, which concerns the continuity of service.
- Mean Time To Failure (MTTF): the expected time that a system will operate before the first failure occurs, which concerns the occurrence of the first failure. Reliability and MTTF are related by the following equation:

$$\mathrm{MTTF} = \int_0^\infty R(t)dt \tag{1}$$

- Availability or $A(t)$: the probability that a system is operating correctly and is available to perform its functions at the instant time t, which concerns the system readiness for the usage. For a simplex system with EDAC, steady-state Availability $(A(\infty))$ is related to MTTF and to Mean Time To Repair (MTTR) by the following equation:

$$A(\infty) = \frac{\mathrm{MTTF}}{\mathrm{MTTR} + \mathrm{MTTF}} \tag{2}$$

$R(t)$ and $A(t)$ can be estimated analytically by using Markov models [43]–[45]. The resulting expressions for systems with scrubbing are a function of t, λ and μ, where λ is the upset rate (approximating SEU by a Poisson process) and μ is the SEU repair rate for configuration memory. For analytical purposes, both λ and μ are assumed to be constant, leading the latter to a MTTR equal to $1/\mu$.

The Availability $(A(\infty))$ expressions for TMR (with correction capability) systems can also be found in [45]. An important consideration from this previous work is that $A(\infty)$ for systems with a fixed scrubbing rate improves (up to the maximum) with increasing scrub period, but after a certain period it drops relatively fast. Therefore, for a given λ and μ, one should choose the scrub period that maximizes $A(\infty)$. Availability numbers based on experimental data are also given in [6].

Assuming that the application needs to be stopped while repairing (i.e., scrubbing), MTTR is lower when scrubbing a single *frame* than when scrubbing the full device, and this improves Availability. However in a TMR system, the application may not need to be stopped as long SEUs are affecting no more than one design module. In that case Availability is not impacted until a second SEU affects a second module.

TMR systems with an arbitrary number of partitions can be implemented to tolerate Multiple Independent Upsets (MIU) in multiple redundant circuits. Markov model theory and experimental validation for such systems is presented in [44], where reliability is shown to improve with increasing number of partitions. Another consideration is the possibility of having persistent errors [46]. Such type of errors cannot be mitigated by scrubbing, and the system becomes permanently unavailable. A Markov model for a TMR system having a fraction of errors being persistent is also presented in [43].

An alternate way to evaluate the reliability is found in [24]. In this work, a reliability model for estimating the MTTF of designs implemented on Virtex-4 XQR4VSX55 FPGA is proposed. The failure probability during a single scrub cycle is analyzed, and the composite failure rate λ_c is the summation of several $\lambda_i \rho_i$ terms. Each term is the probability of being in an orbit condition ρ_i, times the failure rate during such orbit condition λ_i. The obtained expression for MTTF includes two factors that need to be obtained separately by different means. The first factor, which is the probability of having i SEUs during the

scrub period, needs to be estimated using CREME96 orbit-averaged static SEU rates. The second factor, which is the probability of failure during the scrub period if i SEUs occurred during that period, needs to be obtained experimentally. For the latter, two experimental methods are tested and correlated: fault injection and accelerator testing. This experimental work yields a few important conclusions for the system designer:

- The MTTF of a design with TMR and scrubbing is about four orders of magnitude greater than the non-TMR design.
- With the same FPGA and irradiation conditions, the reliability is very design-dependent.
- The scrub rate increase reduces the probability of having more than one SEU per scrub period, and therefore improves the reliability.

As a final consideration, reliability can be improved at design time by analyzing the most sensitive parts of the design and performing a reliability-aware place and route of the design in the FPGA [47], [34].

B. Power Consumption

Configuration memory scrubbing comes at a power cost that can be split in two components: power consumed and dissipated by the configuration port and built-in configuration controller, and power consumed and dissipated by the *scrubber* circuitry. When the *scrubber* is internal to the Virtex FPGA, both components add up and contribute to the overall FPGA power consumption and dissipation. When the *scrubber* is an external device, only considering its static power dissipation makes the overall system power increase substantially.

The first thing to be considered is that power overhead is driven by the scrub or readback rate. If these rates are kept fixed to a value such that reliability requirement is met under peak radiation levels predicted for the mission, most likely the power overhead is unnecessarily high most of the time. However, if rates are dynamically adapted to the radiation environment, the power overhead can be reduced to the the minimum necessary at any given time. The former would be the case for a fixed rate scrubbing methodology, while the latter could be achieved with an adaptive rate methodology, both described in Section VI.C.

Another factor contributing to the power overhead is the configuration clock operation during scrub or readback phases. For SelectMAP (the usual configuration port used for scrubbing), the clock can be free-running or actively controlled (i.e., clocking only when necessary). The latter can be used in order to minimize power due to clock switching during readback and scrubbing, even to keep clock idle during the rest of the time. For example, in a corrective scrubbing scenario (see Section VI.A), variable clocking frequency can be envisaged in order to perform readback at lowest possible frequency (while meeting the readback rate requirement) and to perform scrubbing at higher frequency in order to minimize the MTTR and improve Availability (see Section VIII.A).

In spite of the importance of power consumption at system level, no power characterization for mitigation techniques was found in previous work. This is outlined as a line for future work.

IX. Conclusion

This paper highlights the benefits of SRAM-based FPGAs, and in particular Xilinx Virtex-4QV and -5QV FPGAs, for high-performance reconfigurable applications, and at the same time addresses the inherent susceptibility to ionizing radiation. Virtex FPGA architecture is recalled, conceptually separating Application Layer from Configuration Layer. Radiation effects at both layers are reviewed, focusing on the latter. Configuration memory scrubbing basics, methodologies and architectures are identified and classified, trading-off benefits and limitations for each particular choice. Finally, system-level reliability, availability and power consumption considerations are discussed, as they provide driving requirements to the *scrubber* design. Future work is outlined in the research of novel design methodologies and architectures for reliability-aware and power-aware configuration memory *scrubbers* for SRAM-based FPGAs.

References

[1] J. Wang, R. Katz, J. Sun, B. Cronquist, J. McCollum, T. Speers, and W. Plants, "SRAM based re-programmable FPGA for space applications," *IEEE Trans. Nucl. Sci.*, vol. 46, no. 6, pp. 1728–1735, Dec. 1999.

[2] S. Habinc, "Suitability of Reprogrammable FPGAs in Space Applications," Gaisler Research, Feasibility Rep., 2002.

[3] K. Morris, "FPGAs in space," *FPGA and Programmable Logic J.*, 2004.

[4] B. Osterloh, H. Michalik, S. Habinc, and B. Fiethe, "Dynamic partial reconfiguration in space applications," in *Proc. AHS '09. NASA/ESA Conf. Adaptive Hardware and Syst.*, 2009, pp. 336–343.

[5] R. Roosta, "A comparison of radiation-hard and radiation-tolerant FPGAs for space applications," NASA Electronic Parts and Packaging (NEPP) Program JPL D-31228, 2004.

[6] M. Berg and K. LaBel, "Determining the best-fit FPGA for a space mission: An analysis of cost, SEU sensitivity, and reliability," in *Proc. Microelectron. Reliab. Qualificat. Workshop (MRQW)*, 2007.

[7] C. Carmichael and C. W. Tseng, "Correcting single-event upsets in Virtex-4 FPGA configuration Memory," Xilinx Application Note (XAPP1088), 2009.

[8] Y. C. Wang, "Recommendations for managing the configuration of the RHBD virtex-5QV," Xilinx Presentation in NASA Military and Aerospace Programmable Logic Devices, MAPLD, 2011.

[9] M. Ceschia, M. Violante, M. Reorda, A. Paccagnella, P. Bernardi, M. Rebaudengo, D. Bortolato, M. Bellato, P. Zambolin, and A. Candelori, "Identification and classification of single-event upsets in the configuration memory of SRAM-based FPGAs," *IEEE Trans. Nucl. Sci.*, vol. 50, no. 6, pp. 2088–2094, Dec. 2003.

[10] P. Adell and G. Allen, Assessing and Mitigating Radiation Effects in Xilinx FPGAs. Pasadena, CA, Jet Propulsion Laboratory, California Inst. of Technol., 2008.

[11] G. Allen, G. Swift, and C. Carmichael, VIRTEX-4 VQ Static SEU Characterization Summary. Pasadena, CA, Jet Propulsion Laboratory, NASA, 2008.

[12] G. Allen, Virtex-4VQ Dynamic and Mitigated Single Event Upset Characterization Summary. Pasadena, CA, Jet Propulsion Laboratory, NASA, 2009.

[13] R. Monreal, C. Carmichael, and G. Swift, "Single-event characterization of multi-gigabit transceivers (MGT) in space-grade virtex-5QV field programmable gate arrays (FPGA)," in *Proc. Radiation Effects Data Workshop (REDW)*, Jul. 2011, pp. 1–8.

[14] G. Allen, L. Edmonds, C. W. Tseng, G. Swift, and C. Carmichael, "Single-event upset (SEU) results of embedded error detect and correct enabled block random access memory (block RAM) within the xilinx XQR5VFX130," *IEEE Trans. Nucl. Sci.*, vol. 57, no. 6, pp. 3426–3431, Dec. 2010.

[15] M. Sonza Reorda, L. Sterpone, and M. Violante, "Multiple errors produced by single upsets in FPGA configuration memory: A possible solution," in *Proc. Test Symp., Eur.*, May 2005, pp. 136–141.

[16] H. Quinn, K. Morgan, P. Graham, J. Krone, M. Caffrey, and K. Lundgreen, "Domain crossing errors: Limitations on single device triple-modular redundancy circuits in Xilinx FPGAs," *IEEE Trans. Nucl. Sci.*, vol. 54, no. 6, pp. 2037–2043, Dec. 2007.

[17] H. Quinn, K. Morgan, P. Graham, J. Krone, and M. Caffrey, "A review of Xilinx FPGA architectural reliability concerns from Virtex to Virtex-5," in *Proc. 9th Eur. Conf. Radiation and Its Effects on Compon. Syst., RADECS '07*, Sep. 2007, pp. 1–8.

[18] A. Tylka, J. Adams Jr., P. Boberg, B. Brownstein, W. Dietrich, E. Flueckiger, E. Petersen, M. Shea, D. Smart, and E. Smith, "CREME96: A revision of the cosmic ray effects on micro-electronics code," *IEEE Trans. Nucl. Sci.*, vol. 44, no. 6, pp. 2150–2160, Dec. 1997.

[19] Xilinx Radiation Test Consortium (XRTC), [Online]. Available: http://www.xilinx.com/esp/aerospace-defense/space/xrtc.htm Xilinx Inc.

[20] C. López-Ongil, M. García-Valderas, M. Portela-García, and L. Entrena, "Autonomous fault emulation: A new FPGA-based acceleration system for hardness evaluation," *IEEE Trans. Nucl. Sci.*, vol. 54, no. 1, pp. 252–261, Feb. 2007.

[21] M. Alderighi, F. Casini, S. D'Angelo, M. Mancini, S. Pastore, and G. Sechi, "Evaluation of single event upset mitigation schemes for SRAM based FPGAs using the FLIPPER fault injection platform," in *Proc. 22nd IEEE Int. Symp. Defect and Fault-Tolerance in VLSI Syst., DFT'07*, 2007, pp. 105–113.

[22] M. Aguirre, J. Tombs, V. Baena, F. Muñoz-Chavero, A. Torralba, A. Fernández-León, and F. Tortosa, "FT-UNSHADES: A new system for SEU injection, analysis and diagnostics over post synthesis netlist," in *Proc. NASA Military Aerosp. Programm. Logic Devices, MAPLD*, 2005, p. 2005.

[23] G. Allen, Mitigation Selection and Qualification Recommendations for Xilinx Virtex, Virtex-II, and Virtex-4 Field Programmable Gate Arrays. Pasadena, CA, Jet Propulsion Laboratory, NASA, 2009.

[24] P. Ostler, M. Caffrey, D. Gibelyou, P. Graham, K. Morgan, B. Pratt, H. Quinn, and M. Wirthlin, "SRAM FPGA reliability analysis for harsh radiation environments," *IEEE Trans. Nucl. Sci.*, vol. 56, no. 6, pp. 3519–3526, Dec. 2009.

[25] K. Chapman and L. Jones, SEU Strategies for Virtex-5 Devices Xilinx Application Note (XAPP864), 2009.

[26] H. Quinn, P. Graham, K. Morgan, J. Krone, M. Caffrey, and M. Wirthlin, "An introduction to radiation-induced failure modes and related mitigation methods for Xilinx SRAM FPGAs," in *Proc. Int. Conf. Eng. Reconfigurable Syst. Algorithms,(ERSA08)*, 2008, pp. 139–145.

[27] I. Herrera-Alzu and M. López-Vallejo, "Self-reference scrubber for TMR systems based on xilinx virtex FPGAs," *Integr. Circuit Syst. Design. Power, Timing Modeling, Optimiz., Simulat.*, pp. 133–142, 2011.

[28] D. Johnson, K. Morgan, M. Wirthlin, M. Caffrey, and P. Graham, "Persistent errors in SRAM-based FPGAs," in *7th Annu. Conf. Military Aerosp. Programm. Logic Devices (MAPLD)*, 2004.

[29] M. Berg, C. Poivey, D. Petrick, D. Espinosa, A. Lesea, K. 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," *IEEE Trans. Nucl. Sci.*, vol. 55, no. 4, pp. 2259–2266, Aug. 2008.

[30] M. Berg, "Trading ASIC and FPGA considerations for system insertion," in *Proc. Short Course in Nucl. Space Radiat. Effects Conf., NSREC*, 2009.

[31] A. Vampola, M. Lauriente, D. Wilkinson, J. Allen, and F. Albin, "Single event upsets correlated with environment," *IEEE Trans. Nucl. Sci.*, vol. 41, no. 6, pp. 2383–2388, Dec. 1994.

[32] D. Fay, A. Shye, S. Bhattacharya, D. Connors, and S. Wichmann, "An adaptive fault-tolerant memory system for FPGA-based architectures in the space environment," in *Proc. 2nd NASA/ESA Conf. Adaptive Hardware and Syst. AHS '07*, 2007, pp. 250–257.

[33] A. Jacobs, A. George, and G. Cieslewski, "Reconfigurable fault tolerance: A framework for environmentally adaptive fault mitigation in space," in *Proc. Int. Conf. Field Programm. Logic Applicat., FPL '09*, 2009, pp. 199–204.

[34] C. Bolchini, A. Miele, and C. Sandionigi, "A novel design methodology for implementing reliability-aware systems on SRAM-based FPGAs," *IEEE Trans. Comput.*, vol. 60, no. 12, pp. 1744–1758, Dec. 2011.

[35] J. Heiner, N. Collins, and M. Wirthlin, "Fault tolerant ICAP controller for high-reliable internal scrubbing," in *Proc. IEEE Aerosp. Conf.*, 2008, pp. 1–10.

[36] A. Martin-Ortega, M. Alvarez, S. Esteve, S. Rodriguez, and S. Lopez-Buedo, "Radiation hardening of FPGA-based SoCs through self-reconfiguration and XTMR techniques," in *Proc. IEEE 4th Programm. Logic, Southern Conf.*, 2008, pp. 261–264.

[37] C. Yui, G. Swift, C. Carmichael, R. Koga, and J. George, "SEU mitigation testing of Xilinx Virtex II FPGAs," in *Proc. IEEE Radiation Effects Data Workshop*, 2003, pp. 92–97.

[38] M. Ng and M. Peattie, "Using a microprocessor to configure Xilinx FPGAs via slave serial or SelectMap mode," Xilinx Application Note (XAPP502), 2002.

[39] Y. Li, D. Li, and Z. Wang, "A new approach to detect-mitigate-correct radiation-induced faults for SRAM-based FPGAs in aerospace application," in *Proc. IEEE Nat. Aerosp. Electron. Conf. NAECON '00*, 2000, pp. 588–594.

[40] G. A. Vera, "A programmable configuration scrubber for FPGAs," in *Proc. Military/Aerosp. Programm. Logic Devices (MAPLD) Conf.*, 2009.

[41] M. Gokhale, P. Graham, M. Wirthlin, and D. Johnson, "Dynamic reconfiguration for management of radiation-induced faults in FPGAs," *Int. J. Embedded Syst.*, vol. 2, no. 1, pp. 28–38, 2006.

[42] B. Johnson, *Design & Analysis of Fault Tolerant Digital Systems.* Boston, MA: Addison-Wesley, 1988.

[43] D. McMurtrey, K. Morgan, B. Pratt, and M. Wirthlin, "Estimating TMR reliability on FPGAs using Markov models," Brigham Young Univ. Dept. of Elect. and Comput. Eng., 2007, Tech. Rep.

[44] B. Pratt, M. Caffrey, D. Gibelyou, P. Graham, K. Morgan, and M. Wirthlin, "TMR with more frequent voting for improved FPGA reliability," Brigham Young Univ. Dept. of Elect. and Comput. Eng., 2008, Tech. Rep.

[45] Z. Wang, L. Ding, Z. Yao, H. Guo, H. Zhou, and M. Lv, "The reliability and availability analysis of SEU mitigation techniques in SRAM-based FPGAs," in *Proc. Eur. Conf. Radiat. Its Effects on Compon. Syst. (RADECS)*, 2009, pp. 497–503.

[46] K. Morgan, M. Caffrey, P. Graham, E. Johnson, B. Pratt, and M. Wirthlin, "SEU-induced persistent error propagation in FPGAs," *IEEE Trans. Nucl. Sci.*, vol. 52, no. 6, pp. 2438–2445, Dec. 2005.

[47] L. Sterpone, M. Reorda, and M. Violante, "RoRA: A reliability-oriented place and route algorithm for SRAM-based FPGAs," in *Research in Microelectron. Electron.*, 2005, vol. 1, pp. 173–176.