Area-Driven Partial Reconfiguration for SEU Mitigation on SRAM-based FPGAs

***Abstract*—This paper presents an area-driven Field-Programmable Gate Array (FPGA) scrubbing technique based on partial reconfiguration for Single Event Upset (SEU) mitigation. The proposed method is compared with existing techniques such as blind and on-demand scrubbing on a novel SEU mitigation framework implemented on the ZYNQ platform, supporting various SEU and scrubbing rates. Our solution achieves 22% higher availability on average with 13% less data transfers from a Double Data Rate Type 3 (DDR3) memory when compared with a blind scrubbing approach for a second order polynomial evaluation function. The comparison with an existing on-demand scrubbing technique shows that our approach has 35% lower area with a 4.7% lower availability for the same case study.**

I. Introduction

Recently, the use of FPGAs as implementation platform for various applications as compared to Application-Specific Integrated Circuits (ASICs) has grown, as they offer low non-recurring engineering costs and fast prototyping. More specifically, in applications that operate in a radiation environment, four types of FPGA are usually considered by hardware designers.

Anti-fuse FPGAs are one time programmable and they have inherent tolerance to radiation. Flash based FPGAs have reconfiguration features but they suffer from high total ionizing dose making them inappropriate for long missions [1]. Radiation hardened FPGAs overcome reliability problems, but are more expensive and lack performance as compared to Static Random Access Memory (SRAM) FPGAs for many application domains [2]. SRAM FPGAs offer high performance and reconfigurability, features that are essential for harsh radiation environments. However, the configuration memory of modern SRAM FPGAs is more vulnerable to SEUs as compared to Block Random Access Memorys (BRAMs), and therefore this work targets soft errors only on the configuration layer of an SRAM FPGA [3].

Soft errors caused by radiation particles such as neutrons and protons are not only witnessed in space but also on earth, and in particular in high altitude areas and scientific laboratories such as CERN [3]. Depending on the nature of the application, a strike of a particle on a semiconductor node might cause a soft error that will affect the functionality of the application leading to a system failure. Due to the type of the targeted device (i.e FPGA), transient faults can be removed by reconfiguration, while permanent ones destruct an SRAM configuration memory cell of the fabric, necessitating relocation of the application on another functional part of the device.

Availability, named the percentage of time a system operates correctly over the total time of operation, is a design objective for electronic systems. FPGA-based fault-tolerant systems that require high availability, use redundancy schemes such as Dual Modular Redundancy (DMR) and Triple Modular Redundancy (TMR) to detect and mask faults at the application level and scrubbing, that is the rewriting of the configuration memory of the FPGA device with an error-free bitstream, for fault correction.

When high availability is demanded and power consumption and area is of importance, novel techniques should be investigated for both detection and correction of faults.

The key contributions of this work are:

- an SEU mitigation framework supporting fault injection at the SRAM configuration layer. The framework operates on a single FPGA device (ZYNQ) and uses dynamic reconfiguration techniques to emulate the under protection FPGA part, leading to low-cost solutions.
- a novel scrubbing technique, driven by area information of the under protection part of the Circuit Under Test (CUT), exploring as such the availability vs area trade-off.

II. Related Work

The field of fault-tolerant FPGA-based systems for radiation environments has been extensively researched the last decade. Recent works that represent the state of the art on SEU mitigation techniques and frameworks are summarised in Table I.

A. Existing SEU Mitigation techniques

1) Fault Detection and Masking: The most popular technique for detecting errors is DMR. Two identical modules operate in parallel with a comparator checking their outputs for discrepancies. This technique has 2x area overheads as compared to the original circuit. On the other hand TMR is used as a fault making solution. Three identical modules operate in parallel and their outputs are connected to a majority voter that is able to mask a single error. This technique has 3x area overheads as compared to the original circuit. Both DMR and TMR can be applied at different granularities providing various degrees of fault-tolerance.

2) FPGA scrubbing architectures: Fault recovery is achieved on SRAM FPGAs using scrubbing. The SRAM configuration memory is rewritten in order to prevent accumulation of SEUs. Scrubbing architectures can be categorised based on the portion of area that they reconfigure. Device

TABLE I: Comparison with related work

Related Work	fault detection	scrubbing technique	seu distribution	fault injection	FPGA device	performance
[8]	DMR, TMR	on-demand	N/A	N/A	VII Pro, V4	N/A
[7]	DMR, TMR	blind, on-demand	Poison	custom SEU injector	V5	100MHz
[9]	online checker, TMR	on-demand	N/A	custom SEU injector	V5	N/A
[12]	DMR, TMR	shifted scrubbing	N/A	custom SEU injector	V5	50MHz
[6]	TMR	blind, on-demand, CRC, ECC, SECDED	Gaussian	Xilinx SEU controller	V5	50MHz
[11]	DMR, TMR	on-demand	N/A	N/A	Virtex-5QV	N/A
[4]	TMR	frame-level scrubbing	N/A	custom SEU injector	V5	50MHz
our work	DMR	blind, on-demand, area-driven scrubbing	Any	Xilinx SEM core	ZYNQ	100MHz

scrubbing architectures involve the reconfiguration of the whole FPGA device with an uncorrupted bitstream. Frame-level scrubbing reconfigures one or multiple frames, which are the smallest addressable regions that can be reconfigured on Xilinx SRAM FPGAs. A frame-level redundancy scrubbing method that minimises energy and area overheads compared to other scrubbing techniques was presented in [4].

Another way of grouping scrubbing architectures is based on the location of the scrubbing circuitry. An internal scrubber architecture is located with the rest of system on the same FPGA board and is vulnerable to SEUs but consists a low-cost solution since a single FPGA is used. On the other hand, an external scrubber requires a second device that is usually radiation hardened and initiates the scrubbing of the other device that includes only the application [5].

3) FPGA scrubbing techniques: Three main types of scrubbing techniques are found in literature: blind, readback and on-demand scrubbing. Blind scrubbing is a periodical rewriting of the configuration bitstream based on a user defined scrub rate without having any fault detection awareness. Blind scrubbing has been evaluated in [6] for different SEU and scrubbing rates for an 128-bit AES application. Jacobs et al. [7] proposed blind scrubbing with a user defined rate to prevent accumulation of bit-flips for FPGAs in space.

Readback scrubbing is performed by reading back the configuration bitstream and check for corrupted bits by comparing with a golden copy of the bitstream stored in a protected external memory. In particular, [6] compared Cyclically Redundancy Check (CRC)-based scrubbing with Error Correcting Codes (ECC)-based scrubbing and Single Error Correction and Double Error Detection (SECDED)-based scrubbing using the Xilinx SEU controller, combined with variations of TMR schemes for an 128-bits AES algorithm. The results show that a faster response time to errors as compared to blind scrubbing can be achieved. However, readback of the bitstream and scrubbing triggered after every bit-flip, even if it is not a critical one, leads to increased number of data transfers from the external memory.

On-demand scrubbing is triggered by an error detected by a detection mechanism. An on-demand triggered scrubbing technique was proposed in [8] that offers fast recovery time by exploiting DMR and or TMR at different granularities for fault detection and masking. Straka et al [9] proposed the use of online checkers or TMR for detecting faults and an on-demand scrubbing methodology orchestrated by a partial reconfiguration controller for reconfiguring the faulty module only. A novel scrubbing method that reduces the Mean Time to Repair (MTTR) by 30% based on the exploitation of the non-uniform distribution of the critical bits of the configuration memory, was suggested in [10]. The fault recovery starts by reconfiguring frames that include critical bits and then reconfigures the rest of the frames. A resource efficient mitigation of SEUs on FPGAs used in space application was introduced in [11]. The radiation level is monitored using internal BRAMs on a Xilinx 5QV device and on-demand scrubbing is triggered by an error detection in a TMR or DMR module.

B. Existing SEU Mitigation Frameworks

SEU mitigation techniques are evaluated under state-of-the-art SEU mitigation frameworks proposed in the literature. A reconfigurable fault-tolerant framework for space applications where the level of fault tolerance is controlled based on the radiation levels in several orbits was proposed by [7]. The hardware architecture was implemented on a Xilinx Virtex 5 FPGA running on 100 MHz. The preceding work assumes that the inter-arrival time of particles follows a Poisson distribution. An alternative SEU framework was implemented on a Xilinx ML506 board including a Xilinx Virtex 5 FPGA, where the fault injection process was performed using a custom SEU injection tool through the Joint Test Action Group (JTAG) port. A single FPGA fault injection system implemented on a Virtex 5 FPGA running on 50 MHz and using a custom fault injection tool was proposed in [12]. Brosser et al [6] designed a framework for mitigating radiation effects running on 50 MHz. The assumed SEU distribution was Gaussian and the fault injection was enabled by the use of the Xilinx SEU controller. Finally, the authors in [11], introduced a framework for monitoring radiation using internal BRAMs on a Xilinx 5QV FPGA without mentioning the presumed fault injection method and the performance of operation.

Most of the SEU mitigation frameworks use partial reconfiguration via the Internal Configuration Access Port (ICAP)

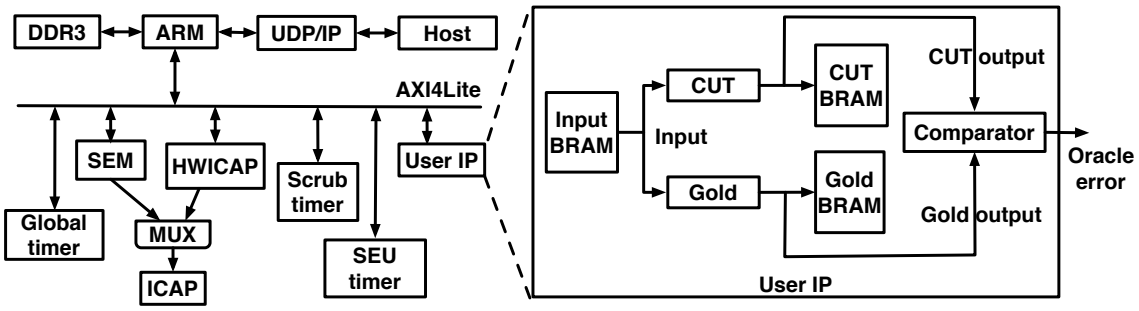

Fig. 1: Hardware prototype

and have been prototyped and evaluated on old FPGA devices. The main difference of this work with the existing approaches are a) the implementation of the SEU mitigation framework on a modern ZYNQ FPGA architecture which allows high design flexibility due to the ARM processor, b) the support of any SEU distribution that mimics the real world for the inter-arrival time of particles and c) the use of a single platform for the emulation of a radiation environment providing as such a low-cost solution to researchers and practitioners.

III. Proposed SEU mitigation method

This work advocates a novel area-driven reconfiguration scheme for fault-tolerant FPGA designs. The motivation behind the proposed technique is the exploitation of the trade-off between availability of the system, and the area and memory transfer requirements. The proposed method is based on DMR for detecting errors, but instead of replicating the whole design under protection, it uses the partial reconfiguration property of modern FPGA devices, and replicates only part of the design at any given time.

Considering a modular component that can be partitioned into smaller modules, each one of the modules are being checked in a cyclic order instead of replicating the whole component. The proposed area-driven technique checks the modules of the component for errors for time that is proportional to their area. Therefore, the module with the largest area is tested for errors more time than the rest of the modules.

IV. The Poroposed SEU Mitigation Framework

The SEU mitigation framework designed in this work supports fault injection at the SRAM configuration memory of the FPGA using the Soft Error Mitigation Core IP (SEM) provided by Xilinx [14]. In addition to the fault injection, FPGA scrubbing is provided through partial reconfiguration via the ICAP port to correct the errors caused by the injected SEUs.

A. Hardware

In this work, the system that is envisioned consists of two parts that communicate with each other with the former being protected while the latter being vulnerable to SEUs. The assumed protected part using TMR is smaller in size than the second part and provides the input vectors and accommodates the partial reconfiguration controller, while the second part holds the actual CUT.

The experimental system consists of a desktop PC and a Zynq FPGA board. The PC is responsible for configuring the parameters of the experiment, such as SEU events and CUT, as well as for retrieving the data from the board and analyse them. The FPGA board consist of the supporting framework, that is the part responsible to inject the SEUs to the configuration memory allocated to CUT, perform the desired SEU mitigation strategy, as well as logging any activity.

Figure 1 shows the block diagram of the hardware architecture implemented on the Zedboard with XC702 FPGA. The host is a Desktop personal computer with an Intel i7 2600 running Matlab. The ARM processor runs the LWIP stack for communicating with the host PC via Ethernet over the UDP/IP protocol [13]. The SEM core is used for injecting the configuration bit-flips using the ICAP port. The ICAP port is shared between the SEM core and the HWICAP module. The Hardware Internal Configuration Access Port (HWICAP) controller performs the scrubbing by transferring the bitstream from the DDR3 to the ICAP port. Three 32-bit AXI timer modules are connected on the AXI4lite bus and five interrupts from the programmable logic (PL) to the processing system (PS) are used for keeping timestamps (time of events) for the experiments. The User IP includes an input BRAM that stores and feeds with input data both the CUT and the Gold designs. Moreover, the CUT and Gold BRAMs are for debugging purposes and a comparator is checking the outputs of the CUT and the Gold circuits.

B. Xilinx Essential Bits

It should be noted that Xilinx offers the essential bit technology that clusters the configuration SRAM bits into essential and not essential for the operation of the CUT. An essential bit, is a bit that if flipped might cause a functional error. A critical bit, is an essential bit that if flipped, produces a functional error. Not all essential bits are critical bits. To find the critical bits of an FPGA-mapped design, one should perform fault injection at the configuration SRAM layer of the FPGA and mark as critical the bits that produce an error at the output of the CUT. In this work, the list of the essential bits is sampled randomly to flip configuration bits that correspond to the CUT and reduces the time of the fault injection campaign as there

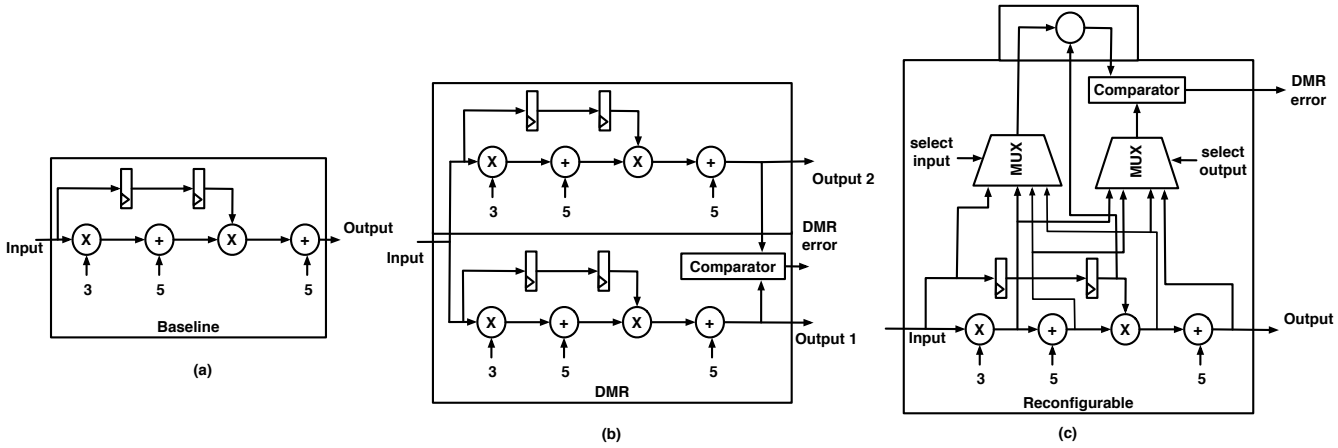

Fig. 2: Three versions of the polynomial case study

is no need to inject faults on the whole FPGA configuration memory.

C. Software

Figure 3 shows the flowchart of the software routine running on the ARM core in bare-metal. The key parts of the software routine are: a) the communication with the host, b) the control of the interrupts and c) the management of the partial reconfiguration process. In more detail, the host PC, transfers the input data for the user IP, the bit-flip addresses, the time intervals of the SEU events, the scrub rate, the duration of the experiment and the partial bitstream for the user IP to the DDR3 memory. The input data are loaded to the input BRAM on the PL. The user IP starts by reading the input data in cyclic way so that it processes always the same data. Then the global timer starts that counts the execution time of the experiment.

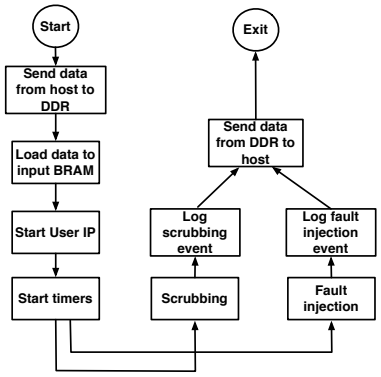

Fig. 3: Flowchart of the Software

When the fault is going to be injected, the CUT pauses its operation and resumes execution after the fault is injected in the system. The reason for this is that the injection latency varies depending on the address of the bit flip. When the fault injection process completes, a time-stamp is logged and stored in the DDR3 memory. In the same way, after a scrubbing process completes, a time-stamp is logged as before. When the global timer reaches the amount of execution time that the user specifies, the interrupts are disabled and the ARM core sends all time-stamps back to the host PC for analysis.

V. Experimental Results

The case study used in this work is a polynomial evaluation function

$$F(x) = 3x^2 + 5x + 5 \quad (1)$$

where the input x is 16-bits. The reason this case study was selected is because it can be partitioned in its arithmetic operators for protection and it is a feed-forward datapath. In addition, the selected case study has common operators found in signal processing applications. Figure 2(a) presents the block diagram the hardware implementation of the aforementioned function. It consists of two adders, one constant coefficient multiplier and a two-inputs multiplier that actually occupies most of the area of the entire circuit. All operators support integer arithmetic with wordlength of 16-bits. The blind scrubbing technique is applied on this design for experimentation.

Figure 2(b) shows the DMR version of the baseline where the entire component is duplicated and a comparator is added for detecting an error. This design is used to evaluate the on-demand triggered scrubbing technique. The proposed solution is shown in Figure 2(c). The design is extended with multiplexers, a comparator that signals a mismatch and a reconfigurable partition for instantiating any part of the design for protection at any given time.

A. SEU Mitigation Framework Results

1) Performance of the framework: The mean SEU rate of the environment the FPGA system will be operating is assumed to be constant over time. The inter-arrival time of particles follows an exponential distribution having a mean that is the constant SEU rate. It should be noted that the framework can support any SEU injection distribution, but in this work it was selected to be used the one from [15].

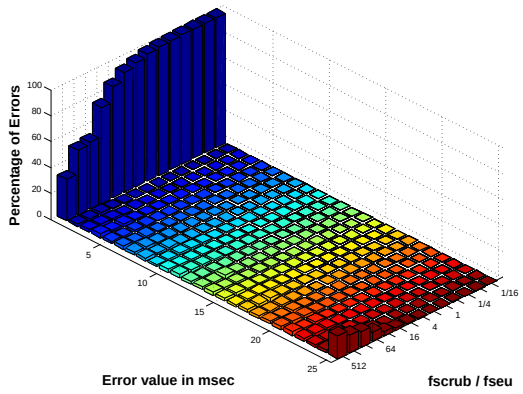

Fig. 4: Performance of the framework

In every experiment, the targeted SEU rate is compared with the actual SEU rate. Figure 4 presents a three-dimensional bar graph of the error value between the actual and the targeted fault injection time versus the scrub over seu rate vs the percentage of errors. It should be noted that increasing the scrub over SEU rate beyond 16 increases the error value between actual and targeted fault injections, deviating from the assumed exponential distribution. The selected SEU rate is 1 SEU per second and if this rate decreases, less errors will be observed, but the actual time of the experiments will be proportionally longer.

Table II includes resources utilization of the framework on the ZYNQ platform without the CUT.

TABLE II: Resources utilization of the framework on the ZYNQ without the CUT

	#LUTs	**# REGs**	**# BRAMs**
SEM core	891	1260	3
HWICAP	386	925	2
AXI4Lite	783	716	N/A
AXI Timers	1,011	648	N/A
User IP (without CUT)	1,471	879	87
Reset logic	23	31	N/A
Total Utilization	4,565	4,459	92
Available	53,200	106,400	140
Percentage(%)	8.6	4.2	65.7

2) Experimental Setup: The SEU rate of the baseline design was 1 SEU per sec per unit area. As unit area in this work it is considered an essential configuration bit. The scrubbing rate is varied from 1/16 to 16 times the SEU rate with an interval of 2. In that way, by increasing the scrubbing rate and keeping the SEU rate fixed, an increase in availability is observed. The on-demand scrubbing method assumes 2 SEUs per sec while the proposed scrubbing technique has a 1.13 SEUs per sec. The SEU rates are calculated based on the number of essential bits of each design. The baseline has 78,652 essential bits as reported by Xilinx tools while the DMR design has 158,381 and the proposed has 100,397.

B. Results of FPGA Scrubbing techniques

The metrics used for evaluating the existing scrubbing techniques and the proposed one are: availability, MTTR, size of data transferred from DDR3 and area.

1) Availability: The availability is the percentage of the time the design is operating producing correct results over the total time of operation. Figure 5 shows the availability of different techniques for several ratios of scrub over SEU rate. The y-axis includes the confidence intervals of the availability for the experiments that were run. The on-demand scrubbing technique for the DMR design achieves the highest availability of 0.99.

Self-aware availability is introduced to indicate whether the CUT has knowledge of when it produces wrong results by checking the DMR error signal from the comparator. The baseline with the blind scrubbing technique does not have any detection mechanism and therefore, self-aware availability is not applicable. The on-demand scrubbing and the area-driven techniques both have self-aware availability.

2) MTTR: The MTTR for the baseline polynomial evaluation function is the time spent for reconfiguration and is 1.189 ms. The DMR design has a MTTR of 2.419 ms while the proposed design's MTTR is 1.599 ms.

3) Size of data transferred from DDR3 for scrubbing: The baseline bitstream size is 17,063 bytes while the size of the bitstream for the DMR is 34,714 bytes. Finally, the reconfigurable design has 17,651 bytes in total with 5,296 bytes corresponding to the area for accommodating the instantiated modules for checking for errors. The size of data transferred from DDR3 depends on the number of scrubs and thus, the count of reads from the DDR3 per technique.

Figure 6 shows the trade-off between unavailability and size of data transferred from DDR3 per technique, having the origin of the diagram as the best performing point. The area-driven scrubbing outperforms blind scrubbing on the unavailability vs data transfer from DDR3 trade-off. However, the on-demand scrubbing method for the DMR design achieves the maximum availability with the lowest memory data transferred.

4) Area: Table III shows area results for the three systems on the ZYNQ platform. The baseline occupies 578 resources (464 Lookup Tables (LUTs) and 114 Registers (REGs) while the DMR's resources utilization is 943 LUTs and 230 REGs (2x the area of the baseline). Our approach saves resources when compared to the DMR version of the system by using 136 REGs and 509 to 622 LUTs, depending on the module that is instantiated on the reconfigurable part of the CUT.

TABLE III: Resources utilization of the CUT

	#LUTs	**# REGs**
Baseline	464	114
DMR	943	230
Proposed	[509,622]	136

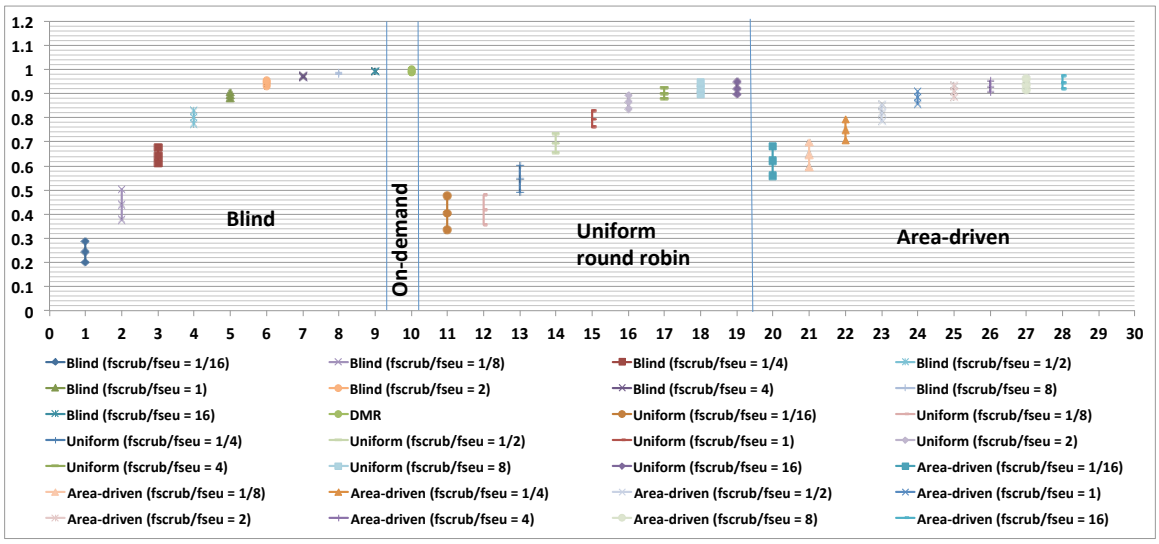

Fig. 5: Availability for different FPGA scrubbing techniques

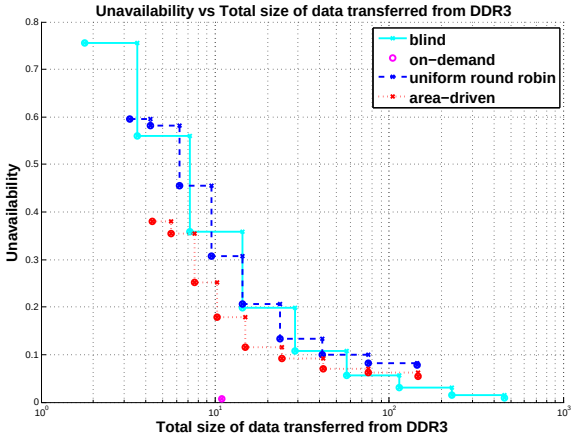

Fig. 6: Unavailability vs Total size of data transferred from DDR3 memory

VI. Conclusion

In conclusion, a new scrubbing technique taking into account area information and using partial reconfiguration is presented in this paper. An investigation with existing scrubbing techniques was enabled by the introduction of a novel SEU mitigation framework. The framework supports different SEU and scrub rates for experimentation on the ZYNQ platform. Experimental results show that our technique saves 35% area when compared to an on-demand scrubbing strategy by achieving 4.7% lower availability. Last but not least, the proposed area-driven technique achieves 22% higher availability on average than blind scrubbing with 13% less data transfers from the DDR3 memory.

References

[1] Felix Siegle, Tanya Vladimirova, Jrgen Ilstad, and Omar Emam. 2015. Mitigation of Radiation Effects in SRAM-Based FPGAs for Space Applications. ACM Comput. Surv. 47, 2, Article 37

[2] I. Herrera-Alzu and M. Lopez-Vallejo, "Design Techniques for Xilinx Virtex FPGA Configuration Memory Scrubbers," in IEEE Transactions on Nuclear Science, vol. 60, no. 1, pp. 376-385, Feb. 2013.

[3] Xilinx Device Reliability Report, UG116 (v10.4) April 1, 2016

[4] J. Tonfat, F. Lima Kastensmidt, P. Rech, R. Reis and H. M. Quinn, "Analyzing the Effectiveness of a Frame-Level Redundancy Scrubbing Technique for SRAM-based FPGAs," in IEEE Transactions on Nuclear Science, vol. 62, no. 6, pp. 3080-3087, Dec. 2015.

[5] M. Berg et al., "Effectiveness of internal vs. external SEU scrubbing mitigation strategies in a Xilinx FPGA: Design, test, and analysis," Radiation and Its Effects on Components and Systems, 2007. RADECS 2007. 9th European Conference on, Deauville, 2007, pp. 1-8.

[6] F. Brosser, E. Milh, V. Geijer and P. Larsson-Edefors, "Assessing scrubbing techniques for Xilinx SRAM-based FPGAs in space applications," Field-Programmable Technology (FPT), 2014 International Conference on, Shanghai, 2014, pp. 296-299.

[7] A. Jacobs, A. D. George and G. Cieslewski, "Reconfigurable fault tolerance: A framework for environmentally adaptive fault mitigation in space," 2009 International Conference on Field Programmable Logic and Applications, Prague, 2009, pp. 199-204.

[8] C. Bolchini, A. Miele and C. Sandionigi, "A Novel Design Methodology for Implementing Reliability-Aware Systems on SRAM-Based FPGAs," in IEEE Transactions on Computers, vol. 60, no. 12, pp. 1744-1758, Dec. 2011.

[9] Martin Straka, Jan Kastil, Zdenek Kotasek, and Lukas Miculka. 2013. Fault tolerant system design and SEU injection based testing. Microprocess. Microsyst. 37, 2 (March 2013), 155-173.

[10] G. L. Nazar, L. P. Santos and L. Carro, "Accelerated FPGA repair through shifted scrubbing," 2013 23rd International Conference on Field programmable Logic and Applications, Porto, 2013, pp. 1-6.

[11] R. Glein, B. Schmidt, F. Rittner, J. Teich and D. Ziener, "A Self-Adaptive SEU Mitigation System for FPGAs with an Internal Block RAM Radiation Particle Sensor," Field-Programmable Custom Computing Machines (FCCM), 2014 IEEE 22nd Annual International Symposium on, Boston, MA, 2014, pp. 251-258.

[12] G. L. Nazar and L. Carro, "Fast single-FPGA fault injection platform," 2012 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), Austin, TX, 2012, pp. 152-157.

[13] A. Suardi, E. C. Kerrigan and G. A. Constantinides, "Fast FPGA prototyping toolbox for embedded optimization," Control Conference (ECC), 2015 European, Linz, 2015, pp. 2589-2594.

[14] Xilinx Soft Error Mitigation Controller v4.1 Product Guide

[15] M. Grecki, "VHDL Simulation considering Single Event Upsets (SEUs)", NSTI-Nanotech 2006, www.nsti.org, ISBN 0-9767985-6-5 Vol. 1, 2006 717