# Performance Analysis of SEE Mitigation Techniques on Zynq Ultrascale+ Hardened Processing Fabrics

Arturo Pérez, Andrés Otero, Eduardo de la Torre
Centre of Industrial Electronics
Universidad Politécnica de Madrid
Madrid, Spain
{arturo,perez; joseandres,otero; eduardo.delatorre}@upm.es

***Abstract*—Zynq UltraScale+ (ZU+) devices contain a variety of computing fabrics of different nature and purpose to support complex adaptive heterogeneous system applications with active fault tolerance techniques. Namely, the Processor Management Unit (PMU) and the R5 processors are candidates for hosting the scrubbing and reconfiguration tasks, as well as for dealing with other fault detection and repair tasks originated in, for instance, Triple-Modular Redundancy (TMR) structures in the reconfigurable logic section. This paper contains a performance and trade-off analysis of different scrubbing techniques (ECC-based and readback) when handled by different components such as the Soft-Error Mitigation IP (SEM-IP), the Real Time Processing Unit (RPU) processors or the Platform Management Unit (PMU) processor. The implementation of the measured strategies has been selected in order to be compatible with dynamic and partial reconfiguration functions in order to also have higher additivity at HW level.**

**Keywords—Zynq UltraScale+, FPGA, fault tolerance, scrubber**

## I. INTRODUCTION

Modern high performance reconfigurable MPSoCs are good candidates to satisfy the requirements imposed by the space revolution of these days. Space missions have been reserved during years to the big space agencies, but nowadays a never seen amount of companies are arising with the goal of exploiting the opportunities that the space market offers. New sensors, as hyperspectral cameras, can expand the capabilities of satellites for science and observation missions. Exploration missions will require fully autonomous operation if they want to travel to really unknown places. Due to these facts the requirements of solutions for space applications are increasing. There is a need of using more powerful devices, in terms of computing power, while keeping reliability, low power, weight and cost. Space agencies have been developing, during the last years, projects for designing new devices to fulfil this gap, as the High Performance Spaceflight Computing (HPSC) Processor initiative of NASA [1], or to test the feasibility of using modern commercial off-the-shelf components (COTS) devices in the space environment, as is the case of the HiRel program from ESA [2].

Another major change is the importance increase of small satellites (SmallSats), as can be seen in the launch plans of Micro/nanosatellites forecast made annually by SpaceWorks [3]. This can be explained because SmallSats significantly decrease the cost of mission development, compare to the traditionally large devices employed, increasing the budget and mission planning flexibility. They can benefit from the last developments in computing devices to implement very capable and flexible solutions to meet the requirements of sensor processing and space applications (size, weight, cost, etc.) with limited resources. The advantages of SmallSats, its trends and biggest challenges are largely explained in [4].

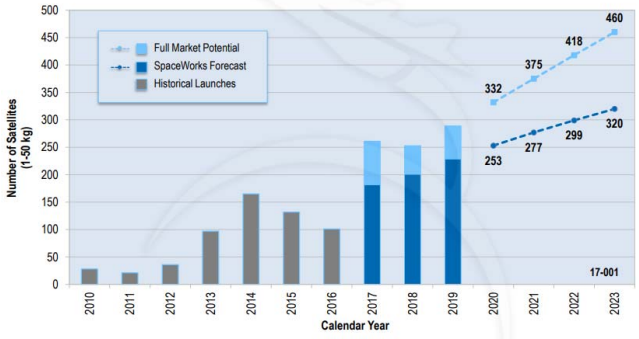


Figure 1 SpaceWorks 2017 Nano/Microsatellite Launch History and Forecast (1 - 50 kg) [3]

These strong requirements on computing power are increasing the interest on COTS devices developed with the most modern technologies in a traditional sector identified by using very reliable, but old technologies. The flagship device of NASA missions during the last decade, the RAD750 processor, can´t fulfil requirements of compute demanding applications with a performance of ~200MOPS. This is pushing the development of new computing platforms based on heterogeneous System on Chips (SoC), which integrate different kinds of computing elements inside a single device, for the space sector, with the examples of the CSP and INNO6FT. Those boards are based on Zynq-7000 devices which integrates a dual-core processor with a reconfigurable FPGA, offering a myriad of solutions, e.g. SMP, HW acceleration, Dynamic Reconfiguration, etc.

This work is focused on the successor family of Zynq-7000 devices developed by Xilinx, the ZU+ [5]. ZU+ is the first truly heterogeneous architecture integrating, in a single chip, symmetric multiprocessors, reconfigurable logic, dedicated peripherals, GPU fabrics and video processors. This combination of elements expand the design possibilities and scalability levels of applications, offering grades of flexibility never seen before in a single device. The reliability of some fabrics has been improved by particular features on its implementations, e.g. modular redundancy, deterministic memory ports, etc. and they are intended to run critical-safety applications. While the application processors, high speed

978-1-5386-7753-7/18/$31.00 ©2010 IEEE

peripherals and video processors have high performance for executing demanding applications. The Programmable Logic offers the flexibility inherent of FPGAs which allow to implement custom solutions for a given application, achieving almost the highest performance, surpassed only by ASIC solutions whose costs are several orders of magnitude higher.

The biggest challenge related to using ZU+ devices in space missions is the presence of ionizing radiation. The effects of space radiation in digital circuits can be mainly classified in: long-term cumulative effects related with the Total Ionizing Dose (TID) or short-term transient effects, commonly known as Single Event Effects (SEE). TID degrades the performance of electronic circuits and is a key factor in the maximum operating life of them. SEE can be subclassified in several categories depending on its effects, but in general, they can be classified either as destructive or non-destructive effects. With regard to reconfigurable SRAM-FPGAs, transient SEE can produce fatal consequences if they affect to the configuration memory of those devices because its functionality can be randomly modified. In this respect, the more relevant SEE are those which produce bit flips in a memory, known as Single Event Upsets (SEU) or Multiple-bit Event Upsets (MEU), depending if they affect to either one or more bits. The configuration memory of ZU+ devices has been characterized in [6] using proton irradiation. This work exposes they can be used on many low earth orbit short duration missions with the appropriate fault mitigation scheme and that its SEE sensitivity has been improved with respect to older families. But as they are not immune to these effects, if ZU+ devices are planned to be used in the space, they will have to be hardened by design techniques, very well studied for SRAM-FPGAs [7], but not sufficiently for reconfigurable MPSoCs [8].

Taking into account the variety and big amount of resources, ZU+ devices have, a lot of possible implementation strategies for fault mitigation. Moreover, some of their hard-wired internal components have been implemented using redundancy, and therefore they have more SEU resilience. The goal of this work is to make an analysis of the performance of these hardened elements to detect and correct SEU using active techniques. Additionally, the embedded features of ZU+ for mitigating SEE have been tested and evaluated. All fault mitigation functionalities implemented have been validated using fault injection through the SEM-IP of Xilinx [9]. Finally, the performance of the fault injection mechanisms are exposed. All these metrics can help to identify and develop the most suitable fault mitigation strategy when designing an application.

The rest of the paper is organized as follows: section II gives an overview of fault mitigation strategies, its possible implementations and, how they can be validated. Sections III explains the main elements of the ZU+ related with this work. The implementation of the fault mitigation techniques developed in this work and the fault injection mechanism used to validate them is explained in section IV. Finally, the performance metrics obtained are shown and analyze in section V. The paper is conclude in section VI.

## II. STATE OF THE ART

Soft errors can affect to almost every digital circuit, affecting execution and producing temporarily wrong results. But in the case of reconfigurable SRAM-FPGAs, soft errors can completely modify the functionality of the design implemented making it useless [10]. For this reason, techniques to prevent, detect and correct soft errors has been well studied on the recent years. Fault mitigation techniques for FPGAs can be developed at different design stages. The fabric logic can be designed in order to have redundancy in the building blocks of the device at different granularity levels, which is the case of radiation tolerant FPGAs. These devices hardened by technology have cost order of magnitude higher than regular, not hardened devices. The robustness of not so hardened devices can be improved at design time by applying passive or active techniques. Passive techniques are usually based either on physical or temporal redundancy. This redundancy can be used to mask faults, avoiding its propagation and the occurrence of failures. Redundancy is also a fast and reliable method to detect errors, but techniques based on redundancy are not able to correct the faults by them-selves, so they have to be combined with active techniques, e.g. scrubbers. Scrubbing is a general term used to refer to methods for correcting upsets in memories. The correction mechanism is to re-write the correct values on the damaged bits. The write process can be done periodically (blind scrubbers), or it can be triggered on-demand when an error has been detected. This is the case of readback scrubbers, one of the backbones of this work, which are periodically reading the memory and checking if any error has occurred. Corrections can be either triggered by redundant modules whose outputs differ, or by information redundancy coding based mechanisms that have detected an error.

Modern MPSoCs which combines, in a single device, hard-wired embedded processors and reconfigurable FPGAs, open up the set of possible fault mitigation techniques to be implemented. Embedded processors are more resilient to soft errors, which can mainly affect its storage elements, but not its implementation. Hence, they are expected to be more robust than the reconfigurable part. Fault mitigation techniques should be combined, exploiting the features of the device where they are implemented. This is the case of the hybrid scrubber strategy developed by A. Stoddard et. Al. [11] which shows significantly low detection and correction times. Furthermore, by combining different elements, each of them with a different nature, diversity is achieved, avoiding same effects on the actors involved in the fault mitigation chain when a SEE occurs. But the fact of combining different elements to perform a given task increase the design complexity because all the elements must be synchronized. Also, the complexity increases even more when real-time constraints have to be applied. The work in [12] shows how different mission tasks can be combined with support functions in a reliable manner, thanks to using a certifiable OS. This work can be used to decide the best global mitigation strategy to implement by combining different elements available in ZU+ devices.

All fault mitigation strategies must be validated. The most realistic method is accelerate irradiation testing, performed in beam-facilities. These tests can expose unexpected effects, as in the case of [13]. However, the high cost of these tests make

them unaffordable for many institutions. Lower-cost solutions have been presented, which are based on fault emulation [14], to generate affordable verification mechanisms. In this work, the proposed fault mitigation strategies have been evaluated using a fast injection procedure, implemented with the SEM-IP and commanded by the R5 cores.

## III. SYSTEM ARCHITECTURE

ZU+ is a really complex device with a variety of diverse computing fabrics. In section I, its main computing elements were briefly introduced. However, there are also several peripherals to interface with different kind of devices, dedicated DMAs to move data efficiently, channels for inter-processor communications, modules dedicated to isolate different elements, etc. This section explains the most relevant elements of the ZU+ architecture related with the work presented in this paper.

### *A. Reliable Fabrics:*

In this paper reliable fabrics refer to those hardened elements within the ZU+ architecture whose reliability has been improved by means of implementing them with particular features. This work has been focused on two of those elements, the RPU and the PMU.

*The RPU* is built up with a pair of real time Cortex-R5F processors which implement the ARM-v7R architecture. They have low interrupt latency, Tightly Coupled Memory (TCM) banks which are low-latency memory that provide predictable instruction execution and predictable data load/store timing. They can run in Symmetric Multiprocessing (SMP) configuration or in dual redundant configuration in lock-step mode. Lock-step is a mode of redundancy where both cores execute the same instructions with the same data but with a time offset of some cycles to avoid common mode faults. The outputs from both cores are compared at every clock cycle and, if they differ, an error is triggered to another entity which will decide the recovery mechanism. They can operate with a single memory because it is protected with ECC coding.

*The PMU* is a dedicated fault-tolerant triple-redundant Microblaze processor without caches. Fault tolerance is achieved by adding redundancy to the processors and ECC codes on RAM interface. It is intended to perform the first boot steps, to manage power and system errors and, in some applications, to run user custom code. Xilinx provides a set of libraries built up upon a standalone basic operating system for implementing several functionalities. It follows a modular approach where the user can decide which modules are enabled or not, depending on the application requirements. It operates with an internal RAM whose size is small, limiting the amount of data it can operate with. PMU RAM is used in the rest of the document to refer to the PMU private memory.

These processors can access the configuration memory through the PCAP interface, explained below.

### *B. Configuration Interfaces – Configuration Security Unit*

Configuration interfaces are ports for accessing the configuration registers of the configuration logic. The configuration logic consists of a packet processor, configuration registers and global signals [15]. All actions involving the configuration of the FPGA are done by writing/reading the configuration registers, and the packet processor controls the flow of data between the configuration interfaces and the configuration registers. The packet processor knows how to perform each action because all data sent to the configuration logic is packed with a set of commands, known as configuration commands. The regular user don't have the need of knowing these commands because they are encapsulated in the *bitstreams* generated by Vivado.

There are several configuration interfaces in the ZU+, which can be classified by the devices (external or internal to the ZU+) that make use of them. This paper focuses on the Processor Configuration Access Port (PCAP) and Internal Configuration Access Port (ICAP) interfaces, since they are placed inside the ZU+ and therefore, have the highest performance and can be used for accessing the configuration memory without needing any additional component. The PCAP can be used by any of the processors inside the ZU+ (RPU, PMU or application processors) and the ICAP allows logic implemented in the FPGA to access the configuration logic (e.g. by a soft-core).

The Configuration Security Unit (CSU) of the Processing System (PS) is the main element of the ZU+ with respect to security. It is composed by two blocks: a secure processor, for secure booting, and the Crypto Interface Block (CIB). The latter has dedicated elements for security (e.g. cryptographic hardware acceleration) and, among other components, the PCAP interface and a dedicated DMA (CSUDMA). The control of which interface is to be used for reconfiguration is managed by writing the CSU registers. The CSUDMA is used to move data between any memory of the system and the configuration logic through the PCAP interface.

### *C. Built-in SEU mitigation features*

To increase robustness of ZU+ against SEE, Xilinx has decided to embed hard-wired logic associated to the PL for continuously checking the presence of soft errors affecting the configuration logic. This logic is based on information redundancy codes which are computed once a configuration is loaded into the configuration memory of the PL, and there are two variants:

*a) Error Correction Codes (ECC),* which is applied at frame level, and it can be used for fault detection and correction of up to 4 bit errors per frame, as stated in [9], and *b) Cyclic Redundancy Check (CRC),* which is applied to the whole configuration memory. It is strong for fault detection but doesn't allow to know where the faults are located.

In older families of Xilinx devices, e.g. Zynq-7000, primitives could be instantiated to make use of these features, as, for instance, the FRAME_ECC primitive [16]. In the ZU+ this is no longer possible and all have to be done through the SEM IP controller, which is explained below.

### D. SEM-IP

It is an IP core designed and distributed by Xilinx which can be used, mainly, for fault detection and correction, for SEU mitigation and to inject faults for verification purposes. It is the only officially supported feature to mitigate SEUs by Xilinx which built a more sophisticated layer over the built-in SEE mitigation circuitry explained above. It expands the capabilities of those embedded features for allowing error classification and error reporting. It uses the ICAP interface for accessing the configuration memory. SEM IP activity is driven by commands. Commands can be sent to the SEM IP using two interfaces: the c*ommand interface* is a simple parallel input for receiving basic commands. It can accept commands for switching between operation modes, insert errors or to perform a SW reset. On the other side, the *monitor interface* is a bidirectional interface, more complex than the command interface. It expands the set of commands that can be used with the SEM IP, adding utility commands, and to retrieve detailed information. It is designed to receive/send information in ASCII format, so a normal solution is to connect it to support logic to expose this interface to the external world through an UART port; allowing to control the SEM IP from outside.

To retrieve information from the controller, two interfaces can be used. The already mentioned monitor interface which prints very detailed information in ASCII format, or the *Status interface***,** which is a set of output pins, each one associated to an operation mode of the SEM IP. The actual operation of the SEM IP can be inferred from the status pins when set to a predefined logic value. Furthermore, there is a heartbeat signal which can be used to check the correct behavior of the SEM IP, which can be affected by soft errors. Two additional status can be obtained from this interface, when all signals are high means that an unrecoverable error has occurred, which can only be fixed by reconfiguring the device; when all signals are low means that the controller is in *idle* state.

The last interface of the SEM-IP relevant to this work is the *CAP interface*. It is used for arbitrating SEM IP access permissions to the configuration interface (ICAP in the case of SEM IP). It is based on three pins that are used for informing which controller has access to the ICAP, or the opposite, to notify it that it has to release the configuration interface. The SEM IP doesn't release the ICAP instantaneously, but a request is performed in order to avoid stopping it while an operation is being performed.

## IV. IMPLEMENTATION AND METHODOLOGY

This section explains the common set-up done for taking all performance measurements explained in section V, how some important challenges for the purpose of this paper have been solved, as well as the scrubber implementation. The development board used to implement the system and to obtain the performance metrics has been the zcu102.

### A. System control and execution

The implementation performed for this work have been based on commands to drive the execution. An UART is used to send the commands to the R5 processors running in lockstep mode, and then they perform the desired action. So all actions are performed on-demand by the user interaction. This makes the RPU the centralized control element which manage the rest of components.

Both ICAP and PCAP are used to access to a single element in the ZU+, the configuration memory, so an arbiter is needed for coordinate the different tasks that use it and avoid collisions. The SEM IP, readback scrubbers and reconfiguration functions are the elements which need access to the configuration memory. Each time the SEM IP has to release the configuration interface, the method explained in [9] is followed.

An Inter Processor Interrupt Channel has been used for communicating the RPU and the PMU. It basically consists on a buffer which is filled with the data desired to be sent. Then, an interrupt is triggered from the sender processor to the receiver one, to indicate that new data is available. The receiver acknowledges the packet by cleaning the interrupt. This method is used to notify the PMU two possible actions: a) to read and check an area of the configuration memory, b) to reconfigure a part of the configuration memory.

### B. SEM-IP implementation

For the work presented here, the SEM IP has been configured for *mitigation and testing* and encapsulated in a custom IP for adding support logic with two purposes: using the command interface with axi4-lite protocol for managing the controller by memory mapped transactions from a processor, on one side, and for performing in HW the logic operations needed to infer the *fatal error* and *idle* status, which will be explained in section III.D.

Additionally, another UART interface has been used for showing SEM-IP's monitor interface to both an external terminal and to the PS. The external connection is used in standard output mode only, to show the results of SEMIP's activity. The connection with the PS is the only interface that send fault location and results of repair to the processors.

### C. Fault injection

Fault injection has been implemented using the SEM IP for testing the performance of the different scrubbers explained in this paper. Two different approaches have been analyzed: a) Using the command interface to drive the injection internally, from the R5 processors; b) Using the monitor interface, to control this task from an external PC where is easy to develop SW to perform this task automatically. In this work it has been done using basic Python scripts.

### D. Information redundancy based scrubber

This scrubber has been implemented using the SEM IP and is based on the built-in SEU mitigation features of the ZU+. The implementation performed is much related to the behavior of the SEM IP. Every time the ICAP is released to the SEM IP, it performs an initialization process where all ECCs, one per configuration memory frame, are computed, and the global CRC is also obtained for the whole FPGA. Those values are stored as golden copies. Then, it enters in observation mode, continuously computing the ECCs and CRC codes and checking them with the golden copies stored in the

initialization process. During this period, if any error is detected, the SEM IP tries to correct it entering in correction mode, and error location information (bit, word, and frame affected) is sent through the monitor interface. If the error can be corrected using ECC, it is automatically done, but if the SEM IP cannot fix the problem, it enters in uncorrectable state and exits correction. When errors are detected only by CRC, the SEM IP enters directly to uncorrectable state.

If the activity of the SEM IP has been stopped to perform a reconfiguration, it will pass through the initialization process. So, if a SEE affects the device during either the reconfiguration or the SEM IP initialization, it won't be detected by the SEM IP. Therefore, in a system which uses HW reconfiguration for switching between different functionalities, a DRP based scrubber is required, at least for this.

### *E. Readback scrubbers implementation.*

Two readback scrubber versions have been implemented, one in the ARM R5 processors (in lockstep mode), and the other in the PMU processor (with TMR). The ZU+ FPGAs have very large configuration memories, due to the large amount of resources available (In the case of the zcu102 it is about 26MB). Hence, time needed to read and compare those amounts of data is high, so both scrubbers have been implemented as partial readback scrubbers. This way, specific areas of the FPGA can be verified, reducing the needed time. This is aligned with the fact that in HW designs there are different components with diverse degree of criticality, so the most critical parts of a design should be checked more frequently.

The general implementation of both readback scrubbers has been performed following the same methodology, which is common with most of readback scrubbers already implemented, but it is platform dependent. It can be divided into three tasks, which are performed sequentially:

*Step 1 - Readback:* configuration memory is read and stored into a memory of the system. Initial frame offset and number of frames to read are passed as input parameters. This step comprises: a) generating a set of configuration commands, which are used to set configuration logic into readback mode; b) send those commands from the main memory to the configuration logic using the CSUDMA; c) to perform another CSUDMA transaction, in the opposite direction, to move the read data to a memory usable by the processors, and d) in some conditions, explained below, all data can't be read in a single CSUDMA transaction. When this happens, the process iterates over the last two steps.

*Step 2 - Comparison*: Data read in step 1 is compared against a golden copy previously stored in the system. A golden copy of the configuration data is stored in a reserved space of main memory at boot time. A mask file used to distinguish between configuration related bits of the configuration memory for a given design, or data associated to memory elements (FFs, BRAMs, etc.), is also stored in main memory at boot time. All configuration bits indicated by the mask file, are compared between the read data and the golden copy, word by word, for all read frames.

*Step 3 – Correction:* Errors detected in the previous step are repaired by reconfiguration. Dynamic Partial Reconfiguration (DPR) is performed at frame level to correct only those frames with errors. Frame-based correction provides low repair times, and so, availability mainly depends only on fault detection time.

This paragraph explains some important considerations related with the ZU+ and the performance of the scrubbers. In some conditions, related with the PCAP operation frequency and the readback destination memory selected, the readback operation may stall and the CSUDMA transaction is never finished. The only control signals that can be used to identify this situation are the interrupts associated to the PCAP in the CSU module. When this happens, an overflow in the read channel of the PCAP interface can be observed. The explanation is that the DMA can't move data as fast as the PCAP is delivering it, and the PCAP operation cannot be interrupted in the middle of a command execution. Hence, the PCAP operating frequency is a key point to work around this issue. The consequence is that only a limited number of frames must be read in a single DMA transaction, and that number is function of the PCAP frequency. Furthermore, this is different if the data is moved from the configuration memory to the external DDR or to the PMU RAM. When the data is moved to the PMU RAM, the performance is much lower, as it can be seen later on Table 1 in the experimental results section. Possible reasons are: a) the performance of the PMU RAM bus is worst and it is not optimized for big data transactions, b) the QoS policies applied to the PMU RAM bus are decreasing its performance. There are no monitors on the PMU RAM bus, as there are in other system buses, implemented using the Axi Performance Monitor IP from Xilinx, so the authors, for the moment, couldn't extract more information from the system to explain this issue. When the CSUDMA transactions are performed from the external memory to the configuration memory (to perform DPR), such problem never appeared, independently of the PCAP frequency. The performance of moving data from the PMU RAM to the configuration memory was not evaluated, because the low size of that memory prevents from storing there golden copies of the FPGA bitstream.

### *F. Time metrics*

All time metrics shown in section V have been obtained by reading a timer of the ZU+ from the processors. The timer is a free running counter working at 100MHz frequency, so the maximum achievable resolution was 10ns. Metrics are taken from the processor involved in each operation, since timer access is instrumented on the same executable code as the one for achieving readback. Although those values would have small errors due to architecture differences, they give an overview of the performance variations obtained by using different solutions. The next two paragraphs show the precise moments when timing has been instrumented:

For readback scrubbers, timing is done just before calling the SW functions and just after returning from them, i.e., for the readback operation, it comprises all time needed to generate the configuration commands, the DMA transactions, and to copy the read data from the auxiliary buffer to the global one.

For the ECC scrubber and fault injection, the status interface pins of the SEM IP were connected to GPIOs of the R5 processors. SW interrupts were associated to those pins and they were triggered when any of those signals changed. The interrupts were used to measure the time required by the SEM IP to monitor the different measured actions, which are: fault injection, fault detection and fault correction. Only one of those actions were monitored on each set of tests, avoiding interferences between them.

## V. Performance analysis

The results obtained are presented in this section. First, the fault injection performance is shown. Second, readback scrubbers performance is analyzed, starting by showing the limitations explained in Section IV.D. Different implementations have been carried out to discuss the readback scrubbers' pros and cons. Correction times of readback scrubbers can be also used for modelling DPR in ZU+. To finish, the performance of the SEM-IP for detecting and correcting faults using ECC codes is briefly analyzed.

### *A. Fault injection through SEM-IP*

Fault injection metrics have been taken when injecting a single fault using the command interface. Error injection latency via the monitor interface is given by Xilinx in [9]. The time obtained via the command interface has been measured in between just before sending the command to the SEM-IP via an axi4-lite transaction, and when the SEM-IP enters in idle mode after the fault injection. It has been checked that on every fault injection the peripheral passes through *injection state*.

For the command interface version, injection time is 16.5us (60606 faults/s) when cache memories are enabled in the R5 processors. When cache memories are not enabled the injection time is 21.5us (46500 faults/s). When using the monitor interface, error injection latency tested on UltraScale+ VU3P at ICAP maximum frequency and assuming no throttling on the Monitor Interface is 81µs [9]. Our implemented command interface version significantly improves fault injection rate of recent works, from 26 to ~60000 per second [14]. However, with this method, faults cannot be injected on every bit of the configuration memory as explained below. This injection method is also faster than using the Monitor Interface.

Faults can only be injected in configuration memory bits which are not masked by the mask file. It means that it only can be inserted in configuration related bits.

### *B. Readback scrubbers*

The number of total frames that can be read in a single transaction is limited by the PCAP operating frequency and the destination memory, as explained in section IV.D. Table 1 shows the maximum amount of frames that can be read in a single CSUDMA transaction, depending on the PCAP frequency and the destination memory.

The maximum amount of configuration frames that can be moved to the DDR in a single CSUDMA transaction is very limited when the PCAP is configured to work at its maximum frequency (actually the maximum PCAP frequency is 200MHz, but the frequency of the default PLL used to drive the PCAP is not multiple of 200MHz). When the PMU RAM is used as the destination memory instead the external DDR, the performance decreased significantly.

TABLE 1 MAXIMUM NUMBER OF FRAMES ALLOWED TO BE READ DEPENDING ON THE DESTINATION MEMORY AND THE PCAP FREQUENCY

| PCAP freq. [MHz] \ Dest. memory | DDR | PMU RAM |
|---|---|---|
| 187.5 | 24 | 2 |
| 150 | 71260 | 2 |
| 125 | 71260 | 3 |
| 93.75 | 71260 | 5 |
| 62.5 | 71260 | 15 |
| 46.88 | 71260 | 30 |

Tables 2, 3 and 4 show the time needed to read the configuration memory depending on the destination memory and the PCAP frequency. Table 5 shows the time needed to check read data against the golden copies. All these tables show the performance differences when tasks are performed by either the R5 processors or the PMU. Different configurations have been tested. The R5 processors were used with and without cache memory. This is interesting because the cross section of the device depends on the area of memory employed in a design [17]. The PMU has been tested with two different implementations: 1) moving the configuration data from the PCAP to the external DDR memory and checking it there. 2) Moving the configuration data from the PCAP to the PMU internal memory. Also in this case, the cross section value depends on the memory used, and is expected to have a higher performance when using the PMU RAM on-chip memory.

TABLE 2 READ TIME WITH PCAP FREQUENCY: 187.5MHZ

| Frames \ Mode | R5 no-cache | R5 cache | PMU DDR | PMU RAM |
|---|---|---|---|---|
| 2 | **65µs** | **18µs** | **99.86µs** | **26.9µs** |
| 5 | **101µs** | **21µs** | **225.95µs** | 74.51µs |
| 15 | **225µs** | **46µs** | **646.64µs** | 193.28µs |
| 30 | 666µs | 136µs | 2.05ms | 312.04 |
| 500 | 6988µs | 1.44ms | 21.52ms | -[a] |
| 5000 | 69.29ms | 14.3ms | 214.36ms | -[a] |
| 50000 | 690ms | 142.95 | 2.14s | -[a] |

[a]. PMU RAM exceeded

TABLE 3 READ TIME WITH PCAP FREQUENCY: 125MHZ

| Frames \ Mode | R5 no-cache | R5 cache | PMU DDR | PMU RAM |
|---|---|---|---|---|
| 2 | **65µs** | **18µs** | **100.98µs** | **27.19µs** |
| 5 | **103µs** | **21µs** | **227.78µs** | 41.02µs |
| 15 | **231µs** | **49µs** | **650.54µs** | 144.89µs |
| 30 | **414µs** | **89µs** | **1.28ms** | 294.72µs |
| 500 | **6.27ms** | **1.35ms** | **21.12ms** | -[a] |
| 5000 | **62.17ms** | **13.36ms** | **210.98ms** | -[a] |
| 50000 | **621.21ms** | **133.48ms** | **2.11s** | -[a] |

[a]. PMU RAM exceeded

Differences between tables 2, 3 and 4 are the operating frequency of the PCAP at which the metrics have been taken. When the total amount of frames to read (left column) was lower than the maximum amount of frames that could be read in those operating conditions, the read process was carried out by reading in a single CSUDMA transaction the total number of frames. If the total amount of frames to read was higher than the maximum number of frames that could be read in those operating conditions, the read process was divided into the

necessary amount of CSUDMA transactions of size equal to the maximum number of frames that could be read per transaction. In the tables, single transactions are marked in bold typeface.

With respect of using the DDR as the destination memory, it is seen that it is better to use the PCAP with lower frequency to move all data in a single transaction. Using cache with the R5 processors increase significantly the performance. Cache memories improve around 4 times the speed at the price of higher cross-section. In all the cases, it is easy to notice the strong linear relationship between the time measured and the number of frames moved, for each evaluated case.

TABLE 4 READ TIME WITH PCAP FREQUENCY: 46.88MHZ

| Mode / Frames | PMU DDR | PMU RAM |
|---|---|---|
| 2 | **105.08µs** | **28.06µs** |
| 5 | **235.66µs** | 42.66µs |
| 15 | **670.87µs** | 90.93µs |
| 30 | **1.32ms** | 164.3µs |
| 500 | **21.74ms** | -[a] |
| 5000 | **217.21ms** | -[a] |
| 50000 | **2.17s** | -[a] |

[a] PMU RAM exceeded

By looking at the previous table, it can be concluded that it is better to move data in big transactions, even when using the PCAP at low frequency.

TABLE 5 COMPARISON TIME

| Mode / Frames | R5 no-cache | R5 cache | PMU DDR | PMU RAM |
|---|---|---|---|---|
| 2 | 109µs | 12µs | 223.4µs | 166.7µs |
| 5 | 270µs | 23µs | 561.4µs | 416.23µs |
| 15 | 808µs | 78µs | 1.67µs | 1.25ms |
| 30 | 1.57ms | 178µs | 3.35ms | 2.5ms |
| 500 | 24.64ms | 3.46ms | 55.7ms | -[a] |
| 5000 | 254.42ms | 34.61ms | 553.7ms | -[a] |
| 50000 | 2.45s | 346.47ms | 5.53s | -[a] |

[a] PMU RAM exceeded

Before analyzing the comparison times it should be said that the R5 has been used working at 500MHz, and the PMU at 200MHz. This difference in speed and in the architecture (being the ARM cores much more advanced that the Microblaze cores), can explain the significant differences in performance found between both processors. The differences are higher when cache memories are used in the R5 cores, achieving a much better performance than in the other modes. There should be more differences in the two modes evaluated with the PMU if the golden copies could be used from the PMU RAM.

In future work, a reliability analysis of the proposed techniques will allow to trade-off between the the different performance results, and hence, the time-to-repair metrics, with the reliability of the fabric in charge of doing the operation.

### C. *Reconfiguration (Correction) time*

Tables 6, 7 and 8 show the time needed to load a new configuration in the device, which can be used to fix bit upsets generated by SEE. Time metrics take into account the command generation, as explained in section IV.F. From the data shown, it is remarkable that the PCAP should be used always at maximum frequency for performing DPR.

TABLE 6 RECONFIGURATION TIME WITH PCAP FREQUENCY: 187.5MHZ

| Mode / Frames | R5 cache | R5 no-cache | PMU |
|---|---|---|---|
| 1 | 10µs | 28µs | 14.1µs |
| 10 | 16µs | 48µs | 18.99µs |
| 100 | 91µs | 220µs | 67.41µs |
| 1000 | 832µs | 1.92ms | 544.95µs |
| 10000 | 8.26ms | 18.91ms | 5.33ms |
| 50000 | 41.25ms | 94.42ms | 26.55ms |

TABLE 7 RECONFIGURATION TIME WITH PCAP FREQUENCY: 93.75MHZ

| Mode / Frames | R5 cache | R5 no-cache | PMU |
|---|---|---|---|
| 1 | 13µs | 35µs | 14.42µs |
| 10 | 17µs | 68µs | 23.63µs |
| 100 | 90µs | 390ms | 113.07µs |
| 1000 | 833µs | 3.6ms | 1.01ms |
| 10000 | 8.26ms | 35.68ms | 9.93ms |
| 50000 | 41.21ms | 178.25ms | 49.61ms |

TABLE 8 RECONFIGURATION TIME WITH PCAP FREQUENCY: 46.88MHZ

| Mode / Frames | R5 cache | R5 no-cache | PMU |
|---|---|---|---|
| 1 | 15µs | 35µs | 16.91µs |
| 10 | 33µs | 77µs | 34.67µs |
| 100 | 238µs | 489µs | 213.28µs |
| 1000 | 2.3ms | 4.59ms | 1.99ms |
| 10000 | 22.88ms | 45.61ms | 19.85ms |
| 50000 | 114.32ms | 227.85ms | 99.21ms |

### D. *ECC scrubber*

Faults cannot be injected on every bit of the configuration memory, and for this reason the first step done to evaluate this mechanism was to select some frames where faults could be injected. This has been done by analyzing which frames were masked by the *mask file,* doing a similar analysis as the explained in [18].

SEM-IP is continuously scanning the configuration memory when it is in *observation state*. In case it encounters an error, it transitions through *detection* and *correction states* and then return to observation state. If another error is found by the device, all transitions through the different states are repeated.

The procedure performed to obtain the metrics is the following. First the SEM-IP was commanded to enter in idle mode. In that mode, it can accept error injection commands. All errors were always injected in the same bit position on every frame used in the tests. Then the controller was commanded to enter in *observation state*, and it directly detected and corrected all errors. Detection times were taken between the moments when transitioning to observation state and when it entered in detection state. Correction times were taken from transitioning to correction state until the controller returns to observation state. As the transitions between states are performed automatically, all time metrics are taken in a single test. Total detection and correction time for these 5 injected faults is 168.32ms, i.e., 33.66 ms in average per fault. Tests have been performed with the SEM-IP running at 100MHz.

Table 10 shows the times to detect and correct errors injected in two separated frames.

TABLE 9 ECC DETECTION/CORRECTION TIME, 5 SEPARATED FRAMES

| Mode \ LFA[a] | 2482 | 12482 | 22482 | 32482 | 42482 |
|---|---|---|---|---|---|
| Detection [ms] | 8.78 | 19.07 | 29.05 | 38.33 | 47.64 |
| Correction[ms] | 5.09 | 5.09 | 5.09 | 5.09 | 5.09 |

a, LFA: Linear Frame Address

TABLE 10 ECC DETECTION/CORRECTION TIME, TWO SEPARATED FRAMES

| Mode \ LFA[a] | 2482 | 42482 |
|---|---|---|
| Detection [ms] | 43.391 | 47.61 |
| Correction[ms] | 5.09 | 5.09 |

a, LFA: Linear Frame Address

Due to the low determinism of SEM-IP detection time, especially when considering it works in a continuous loop, the most pessimistic value should be used to model its behavior. Performance presented varies with respect to the performance metrics given by Xilinx in [9] in approximately 20ms. This could be partly explained by the procedure followed to obtain the metrics. However, the given detection times can be used to know the minimum necessary time needed to trigger a recovery action in the R5 processors.

## VI. CONCLUSIONS

This work presents a performance analysis of the most interesting features available in ZU+ devices for fault mitigation with scrubbers. Two readback scrubbers and one based on information redundancy coding have been implemented. The scrubbers have been analyzed independently, to show its limitations and strengths. The R5 processors perform better, even with deactivated caches, although it should also be considered that the R5 processors might possibly be also used for other repetitive (e.g. rate monotonic) real time mission tasks (not related with fault tolerance or other support functions). The next step is to generate a global mitigation strategy, by integrating some of them, in order to build a robust system able to face every possible dangerous situation generated by the occurrence of SEE. Furthermore, as each scrubber is implemented on a different fabric, by combining them, diversity in HW is obtained. Diversity is a very desirable feature in a global fault mitigation solution, because not all the elements are affected in the same manner by ionized particles. However, challenges are going to appear if the scrubbers are going to be integrated, configuration interfaces arbitration, scheduling, etc. It should be remarked that the design implemented in the FPGA must have redundancy for fault masking, to avoid transient erroneous results. Finally, a fast embedded fault injection mechanism has been implemented, whose performance and limitation have been explained. The fault injection has been used to validate all the proposed scrubbers.

## ACKNOWLEDGMENT

This work has been partially supported by the Enable-S3 project that has received funding from the ECSEL Joint Undertaking under grant agreement No 692455. This joint undertaking receives support from the European Union's HORIZON 2020 research and innovation programme and Austria, Denmark, Germany, Finland, Czech Republic, Italy, Spain, Portugal, Poland, Ireland, Belgium, France, Netherlands, United Kingdom, Slovakia and Norway. This work has also been supported by the Spanish R&D State Plan under grant PCIN-2015-254.

## REFERENCES

[1] R. Doyle, W. Powell, G. Mounce, M. Goforth, S. Horan, and M. Lowry, "High performance spaceflight computing (HPSC) nextgeneration space processor (NGSP) a joint investment of NASA and AFRL," in Proc. Int. Symp. Artif. Intell., Robot. Autom. Space (i-SAIRAS), Montreal, Canada, Juen. 2014

[2] S. Esposito et al., "COTS-Based High-Performance Computing for Space Applications," in IEEE Transactions on Nuclear Science, vol. 62, no. 6, pp. 2687-2694, Dec. 2015.

[3] ] B. Doncaster, C. Williams, and J. Shulman (2017). Nano/Microsatellite Market Forecast. [Online]. Available: http://spaceworksforecast.com/docs/SpaceWorks_Nano_Microsatellite_Market_ Forecast_2017.pdf

[4] A. D. George and C. M. Wilson, "Onboard Processing With Hybrid and Reconfigurable Computing on Small Satellites," in Proc. of the IEEE, vol. 106, no. 3, pp. 458-470, March 2018.

[5] "Zynq UltraScale+ Device Technical Reference Manual," Xilinx, San Jose, CA, USA, UG1085, Dec 2017.

[6] D. M. Hiemstra, V. Kirischian and J. Brelski, "Single Event Upset Characterization of the Zynq UltraScale+ MPSoC Using Proton Irradiation," 2017 IEEE Radiation Effects Data Workshop (REDW), New Orleans, LA, 2017, pp. 1-4.

[7] 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.

[8] L. A. Tambara, P. Rech, E. Chielle, J. Tonfat and F. L. Kastensmidt, "Analyzing the Impact of Radiation-Induced Failures in Programmable SoCs," in IEEE Transactions on Nuclear Science, vol. 63, no. 4, pp. 2217-2224, Aug. 2016.

[9] "UltraScale Architecture Soft Error Mitigation Controller v3.1," Xilinx, San Jose, CA, USA, PG187, April 2018.

[10] J. R. Schwank, M. R. Shaneyfelt and P. E. Dodd, "Radiation Hardness Assurance Testing of Microelectronic Devices and Integrated Circuits: Radiation Environments, Physical Mechanisms, and Foundations for Hardness Assurance," in *IEEE Transactions on Nuclear Science*, vol. 60, no. 3, pp. 2074-2100, June 2013.

[11] A. Stoddard, A. Gruwell, P. Zabriskie and M. J. Wirthlin, "A Hybrid Approach to FPGA Configuration Scrubbing," in *IEEE Transactions on Nuclear Science*, vol. 64, no. 1, pp. 497-503, Jan. 2017.

[12] A. Pérez, L. Suriano, A. Otero and E. de la Torre, "Dynamic reconfiguration under RTEMS for fault mitigation and functional adaptation in SRAM-based SoPCs for space systems," *2017 NASA/ESA Conf. on Adaptive Hardware and Systems (AHS)*, Pasadena, CA, 2017, pp. 40-47.

[13] D. S. Lee, M. Wirthlin, G. Swift and A. C. Le, "Single-Event Characterization of the 28 nm Xilinx Kintex-7 Field-Programmable Gate Array under Heavy Ion Irradiation," *2014 IEEE Radiation Effects Data Workshop (REDW)*, Paris, 2014, pp. 1-5.

[14] Igor Villalta, Unai Bidarte, Julen Gómez-Cornejo, Jaime Jiménez, Jesús Lázaro, SEU emulation in industrial SoCs combining microprocessor and FPGA, Reliability Engineering & System Safety, Volume 170, 2018

[15] "UltraScale Architecture Configuration User Guide," Xilinx, San Jose, CA, USA, UG570, April 2018.

[16] "Vivado Design Suite 7 Series FPGA and Zynq-7000 All Programmable SoC Libraries Guide," Xilinx, San Jose, CA, USA, UG953, Sep 2016.

[17] L. A. Tambara *et al.*, "Heavy Ions Induced Single Event Upsets Testing of the 28 nm Xilinx Zynq-7000 All Programmable SoC," *2015 IEEE Radiation Effects Data Workshop (REDW)*, Boston, MA, 2015, pp. 1-6.

[18] R. Giordano, S. Perrella, V. Izzo, G. Milluzzo and A. Aloisio, "Redundant-Configuration Scrubbing of SRAM-Based FPGAs," in IEEE Trans on Nuclear Science, vol. 64, no. 9, pp. 2497-2504, Sept. 2017.