Complex Upset Mitigation Applied to a Re-Configurable Embedded Processor

Sana Rezgui, Gary Swift, Kevin Somervill, Jeffrey George, Carl Carmichael, and Gregory Allen

***Abstract*—Soft-core processors implemented in static random access memory-based field-programmable-gate-arrays, while attractive to spacecraft designers, require upset mitigation. We investigate a proposed solution involving two levels of scrubbing plus triple modular redundancy and measure its in-beam performance.**

***Index Terms*—Embedded processors, field-programmable-gate-arrays (FPGAs), radiation testing.**

I. Introduction

THE growing interest in using embedded system applications built on static random access memory based field-programmable-gate-arrays (S-FPGAs) in space and the well-known upset sensitivity of S-FPGAs in radiation environment, have led to many research activities for the evaluation of the reliability of their operation in orbit. Two major concerns have been addressed: the characterization of the FPGA sensitivity in beam and the evaluation of the proposed mitigation solution for embedded designs in such a circuit.

Prior work in this area involved testing of a hardened 8051 micro-controller using Hamming code [1] on two programmable logic devices (PLDs). Only the PLD, which implemented the protected registers, was exposed to the beam. The experimenters obtained complete SEU immunity of the protected design while running a 6×6 matrix multiplication program under heavy ion beams. The protected design used more flip-flops (164%) and more logic blocks (184%) compared to the single-string version of the design while the maximum frequency was reduced from 33 to 4 MHz.

Another experiment implemented a similar 8051 micro-controller core on a Xilinx XCV300 FPGA using triple module redundancy (TMR) techniques [2]. The testing used fault injection techniques on an emulation board to deliberately inject errors into the configuration bitstream. The TMR 8051 design used about 4 times more configuration logic blocks (CLBs) and 3.6 times more flip-flops with a maximum frequency of 10 MHz. While this method had a higher hardware overhead compared to the Hamming code approach, it gave a significant improvement in speed. It also provides protection against single event transients (SETs) that is not available in the former approach.

These prior experiments demonstrate the need for a general approach to mitigating transient faults in user logic while reducing pin count, area, and power dissipation. This paper evaluates the effectiveness of a Xilinx tool for creating a fully mitigated embedded systems processor run in a radiation environment. This automated tool employs the triple modular redundancy (TMR) technique to mitigate any design implemented on a Xilinx Virtex FPGA [3], [4]. For this purpose, the Xilinx MicroBlaze\texttrademark intellectual property (IP) soft core was tested under irradiation while running an integer-based fast Fourier transform (FFT) program by the North American SEE Consortium. We report the proton-induced error cross section for the mitigated design as well as detail the additional FPGA resources required as compared to that of the unmitigated (or single-string version).

II. Experimental Approach and Target Processor

Radiation testing was based on the consortium's dynamic characterization apparatus [3] with a Virtex-II XQR2 V6000 as the device-under-test (DUT). This device contains 144 18-Kbit block RAMs, 824 I/Os and 16 395 508 configuration bits. An HM5 225 165 B SDRAM memory was added to the board for external code storage. The test board also hosted a second FPGA to perform two separate functions: 1) configuration readback and scrubbing of the DUT and 2) control and monitoring of the functional operation of the MicroBlaze running the FFT program. Configuration scrubbing is the transparent process of reloading the configuration bitstream to correct upsets. Here "transparent" means that normal device operation runs concurrently and without interruption. The former, the configuration monitor, also detects single event functional interrupts (SEFIs) and automatically reconfigures the DUT when they occur. The configuration scrubbing is done at 16 MHz completing approximately four scrub-cycles per second. The number of readback errors and SEFIs are logged continuously on a host computer. The latter function, the functionality monitor, continuously detects and counts the number of errors in the DUTs outputs. Any mismatches detected by the service FPGA are sent to a separate host computer via custom Visual Basic software. More details

Manuscript received July 15, 2005. This work was supported by the NASA Electronic Parts and Packaging Program (NEPP) and Exploration System Research and Technology (ESR&T). Some research presented here was carried out by the Jet Propulsion Laboratory, California Institute of Technology, under contract with the National Aeronautics and Space Administration (NASA) Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not constitute or imply its endorsement by the United States Government or the Jet Propulsion Laboratory, California Institute of Technology.

S. Rezgui and C. Carmichael are with Xilinx, Inc., San Jose, CA 95124 USA (e-mail: sana.rezgui@xilinx.com).

G. M. Swift and G. Allen are with the Jet propulsion Laboratory/California Institute of Technology, Pasadena, CA 91109 USA (e-mail: gary.m.swift@jpl.nasa.gov).

K. Somervill is with NASA Langley Research Center, Electronic Systems Branch, Hampton, VA 23681-2199 USA (e-mail: kevin.m.somervill@nasa.gov).

J. George is with the Aerospace Corporation, El Segundo, CA 90245 USA (e-mail: Jeffrey.s.george@aero.org).

Digital Object Identifier 10.1109/TNS.2005.860743

about this testing methodology are given in [3]. The only part of the setup that is modified for a new test design is the internal architecture of the functional monitor. Fig. 1 shows the test setup in the chamber at the cyclotron of Texas A&M University.

A. MicroBlaze Soft IP Internal Architecture

The MicroBlaze soft IP core is a reduced instruction set computer (RISC) optimized for implementation in Xilinx FPGAs. This soft IP processor is 32-bit instruction word Harvard architecture with a single-issue pipeline. As shown in Fig. 2, it includes thirty-two 32-bit general-purpose registers (GPRs) and can be configured with two caches for enhanced performance. It supports two data buses: IBM's on-chip peripheral bus (OPB) and an local memory bus (LMB). To communicate with its external peripherals, the MicroBlaze uses a specific bus called general purpose input output (GPIO). Furthermore, it is enhanced with hardware exception handling circuitry as well as hardware debug logic.

B. DUT Test Design

Fig. 3 illustrates the initial MicroBlaze architecture and its connections to the internal memory (internal block RAM) through the OPB and block random access memory (BRAM) buses as well as to the external SDRAM memory. The DUT I/Os are used for a bi-directional data transfer bus between the DUT and service FPGAs, handshaking for data transfer operations, and detection of external interrupt or internal hardware exceptions (unaligned access, illegal op-code, illegal address of the code instruction on the OPB bus (IOPB), illegal address of the program data on the OPB bus (DOPB) and division by zero) in the MicroBlaze soft processor.

The architecture presented in Fig. 3 is a common hardware design for each MicroBlaze program selected to run under the beam. The program code resides in the internal BRAM memory or in external storage transferred to the SDRAM by a loader program. The code is loaded each time the DUT is configured. The MicroBlaze clock speed is set at 33 MHz while the GPIO data transfer rate is set at 250 KHz, to allow the MicroBlaze to process the data requests on the GPIO bus.

C. DUT Test Software

An integer-based Fast Fourier Transform (FFT) [7] was selected as the test program. This program is stored in the internal BRAMs each time the DUT is configured. A fixed set of data is sent from the service FPGA to the DUTs and stored in the internal BRAM memories. The results of the FFT program are returned to the service FPGA and compared to the expected result.

Upon reset, a synchronization sequence is initiated using handshaking signals between the MicroBlaze processor and the service FPGA. Next, the service FPGA sends the DUT 16 complex numbers, each a pair of 16 bit integers. Once the DUT has received all 32 words, it computes the FFT and returns the result to the service FPGA. There the calculated results are compared with the expected values and any errors are reported via the host interface. On the host computer, erroneous results are displayed and recorded in a time-stamped strip chart along with counts of any processor exceptions which must be due to beam-induced upsets or transients.

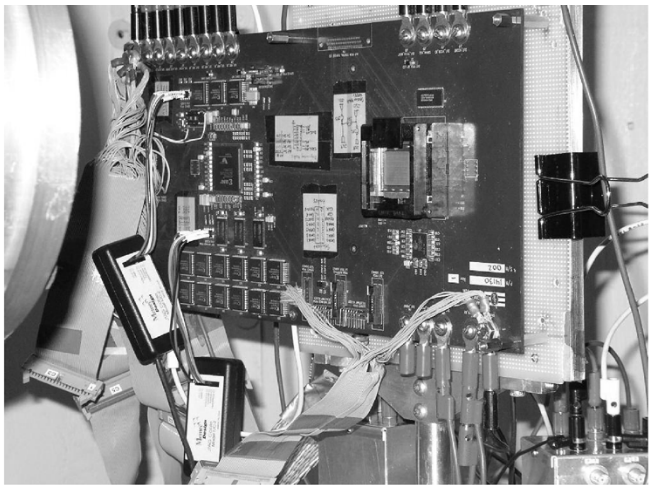

Fig. 1. Test Setup at the Cyclotron of Texas A&M University.

For exception fault detection, a new mechanism has been added to both FPGAs' designs (service and DUT). Indeed, upon detection of any of the five hardware exceptions, the MicroBlaze jumps to a specific address to run the exception's subroutine. By means of minimal software modification of each of the five exceptions' subroutines, the DUT sends a signal to the service FPGA which should acknowledge the detection of this exception to the DUT and to the functional monitor. The DUT pursues then the execution of the program where it was interrupted. In the case of an exception's occurrence (for instance due to an illegal op-code) and if the BRAM is not refreshed, the program will continuously jump to this exception subroutine and it won't be possible to restart normal operation unless correction of the code. It should also be mentioned that not all the possible anomalies of the MicroBlaze operation are detected by the hardware exceptions (for instance some of the illegal instructions do not invoke the illegal op-code exception).

III. Processor Upset Mitigation

The proposed mitigation solution uses the Xilinx TMR tool [8] to triplicate the user design resources, remove half-latches and SRL16 modules, and provide for configuration scrubbing of the DUT internal resources to prevent the accumulation of induced upsets. In addition, the processor's register file, originally implemented using look-up table (LUT)-random access memory (RAM) resources, must be re-implemented using functionally equivalent storage based on user flip-flops (LUT-RAM) storage which should be over-written by configuration scrubbing. Finally, a BRAM scrubbing engine is needed to prevent the corruption of the code or data stored there. For this purpose, one BRAM port must be dedicated to error detection and correction that will leave only one port for the MicroBlaze use, which requires the insertion of the BRAM on the OPB port but not on the LMB port. To allow continuous refreshing of the BRAM contents, a counter is used to cycle through the memory addresses incrementing the BRAM address of the second port and in case the first port of the BRAM is not being used, it rewrites the BRAM content at this specific address with the voted value

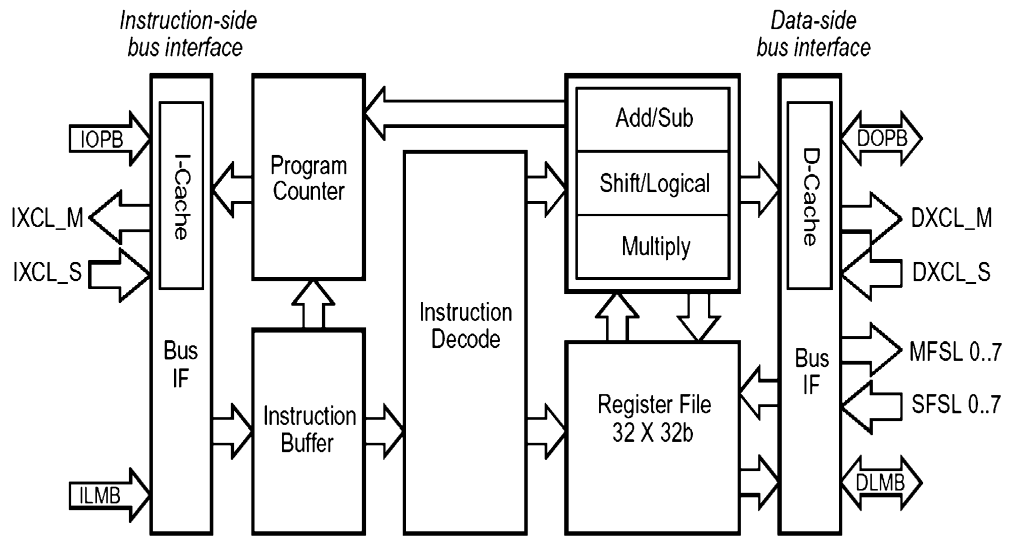

Fig. 2. MicroBlaze core block diagram [6].

from the associated voter (TRV16 in Fig. 4). In case it is used, it skipped this address. Fig. 4 shows the BRAM scrubber block diagram and more details are given in the [9].

Two mitigated versions of the MicroBlaze design architecture have been implemented and tested: with and without the BRAM scrubber. At each of the mitigation phases described above, the internally used resources are counted and displayed in Table I. The given data show the clear hardware overhead in terms of FFs and mainly in terms of LUTs between the two phases 1 (9%) and 2 (47%). This is clearly due to the replacement of the dual-port RAMs. A better idea would be to replace them with BRAM. Nevertheless, it should be noticed here that the overall hardware overhead is less than 40% and is still acceptable for large configuration devices as the Virtex II 6000. Also, the current consumption of the final mitigated design (phase 3) is 0.99 A, which is 2.6 times higher than the initial current value for the single-string design. On the other side, although the mitigated MicroBlaze design has been exercised only at 33 MHz during these radiation testing experiments, the maximum frequency of the mitigated design could run at a frequency up to 66 MHz compared to 77 MHz for the single-string design. For clarity purposes in the following of this paper, the design where the BRAM was not scrubbed (phase 2), will be called *Design 1*, while the fully mitigated design is called *Design 2*.

IV. Proton Test Results

The radiation experiments were conducted, at the Crocker Nuclear Laboratory at University of California at Davis (UCD) using a proton beam of 63.3 MeV. No beam testing was performed on the single-string design. Indeed, it was expected that the error cross-section would be high and the differentiation between the error signatures would not be possible mainly because of the error propagation that could occur between two scrub cycles, which effects won't be corrected by the FPGAs scrubbing. In addition, the code is stored in the internal memories of the FPGA and if not mitigated (not tripled or scrubbed); the code corruption will definitely make the obtained results hard to explain.

Both versions of the mitigated design, with and without the BRAM scrubbing functionality, have been tested (designs 1 and 2). In the case of the fully mitigated design (enhanced with the refreshing of the BRAM contents), the experiments were performed at three widely spaced flux levels to study the effectiveness of the proposed mitigation technique. It is well known that upset mitigation techniques' effectiveness is strongly dependent on the underlying upset rate and thus the flux of the beam or the space environment. For example, the word error-rate for memory arrays mitigated with Hamming codes is approximately proportional to the square of the underlying bit upset rate. Varying the beam flux allows checking this dependence for the more complex mitigation implemented here.

The results are reported in Tables II and III. As shown in column two, the configuration monitor showed that the mitigated design is accumulating an average of five upsets per scrub cycle at a flux of 1.7×10^7 protons/s/cm2. Each time the flux is increased by an order of magnitude, the rate of accumulated upsets per scrub cycle also increases tenfold approximately. The detected errors on the mitigated design were classified in three major types. Type 1 errors were those in which the FFT outputs were wrong. These are subdivided such that they were either corrected after a configuration scrub cycle (type 1a), or they were not corrected after a scrub cycle, even after a reset of the DUT design (type 1b). Type 2 errors were those attributed to nonresponsiveness of the DUT, requiring a reset and synchronization between DUT and service FPGAs that was either corrected by scrubbing and hence referred to as a recovering reset (type 2a), or was not corrected by scrubbing and referred to as a runaway reset (type 2b). This type of error (runaway reset) is an uncorrected error condition that causes the functional monitor to continually attempt to reset the MicroBlaze processor each time the watchdog timer set for the handshaking between the two FPGAs reaches its limit value. Type 3 errors were the occurrence of an exception or interrupt detection.

As shown in Tables II and III, the design's error cross sections increase with the flux. Indeed, if the DUT is exposed to

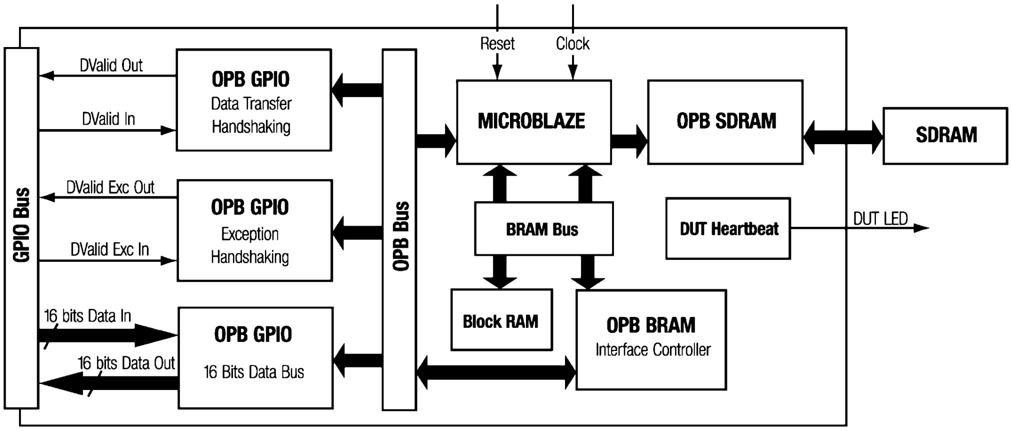

Fig. 3. Block diagram of the microblaze single-string design (before functional triplication).

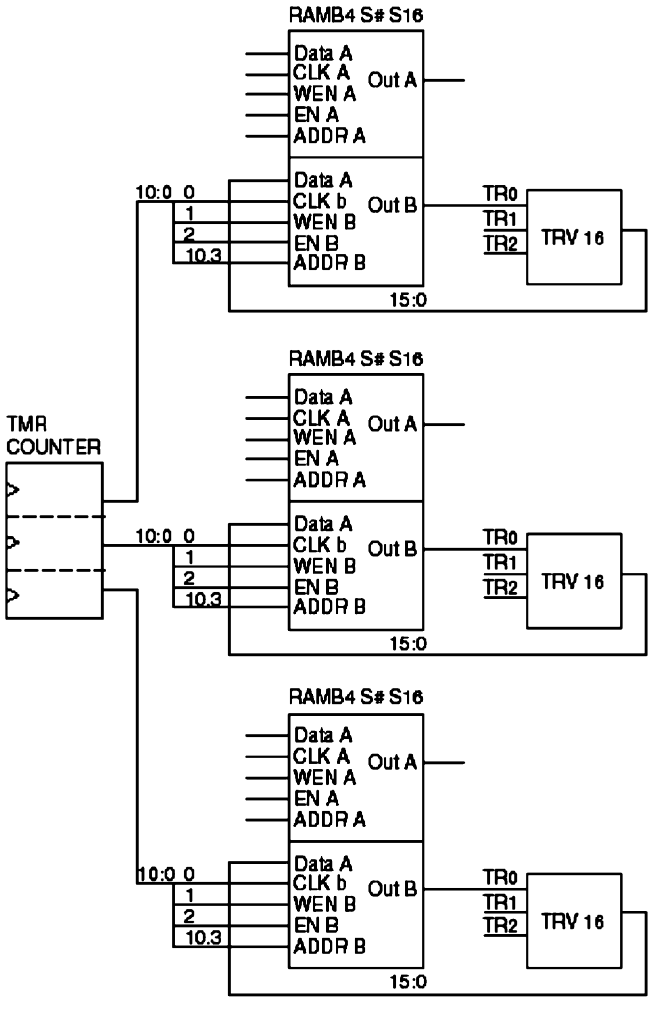

Fig. 4. BRAM mitigation block diagram.

higher number of upsets per scrub cycle then the first rule of the TMR mitigation technique, which fully guarantees the mitigation of less or equal to 1 upset per scrub cycle, is violated and some errors will be expected. Nevertheless, the cross-sections were very low particularly for the fully mitigated design. Indeed, in the case of the *Design 1* where the BRAM is not scrubbed, the five classified type of errors have been observed at different fluxes. On the other hand, in the case of the fully mitigated design, while very few type 1a errors were observed (7 in the case of the experiment performed at the 1.70×10^7 p/cm2/s). Five resets were applied to successfully recovering the DUT from type 2a errors (flux 1) and no errors of type 1b or 2b that would require a reconfiguration of the DUT were observed.

The other error types (types 1b, 2b, and 3) were observed with an increase of the flux resulting in an automatic increase in the upset rate per scrub cycle. While only one error from type 1b has been counted, several errors of type 2b were observed. The errors of type 3 (detected exception) were followed instantly by a runaway reset. Note that at the same flux, the device error cross-section for the *Design 1* ($\sim 1.6 \times 10^{-9}$) is approximately higher of an order of magnitude from the *Design 2* error cross-section ($\sim 1.2 \times 10^{-10}$). As the only difference between both designs is the scrubbing of the BRAMs, the difference between the two cross-sections should be due to the modification of the BRAMs' content and the accumulation of errors. This proves that the continuous refreshing of the BRAMs' content (data and code) was effective.

These results show also that the cross-sections of runaway resets (corrected by reconfiguration of the DUT) are more frequent in the case of *Design 1* and occur only at a high flux in the case of *Design 2*. Indeed, in the case of *Design 1*, since the BRAM is not refreshed the code corruption could be one of the reasons of the existence of these exceptions and runaway resets. While in the case of the fully mitigated design, the increase of flux is expected to overwhelm the TMR mitigation technique although the BRAM is continuously refreshed. Besides, the program code could be corrupted directly by SEUs on the BRAMs or indirectly by SEUs on the CLBs used for the address, data and control busses of the BRAMs or the voters. Hence, if the number of upsets per second (flux) therefore per scrub cycle increases, wrong values could be written by the BRAM scrubber voters.

To make sure that the BRAM code corruption is likely to be the cause of these runaway resets, the BRAM mitigation design has been implemented in standalone mode and tested under proton beams at similar fluxes and at the same facility (UCD). Better results from the BRAM mitigation design in standalone mode are expected than when used in the MicroBlaze design for two main reasons. First, in standalone mode, the BRAM data is

TABLE I
COMPARISON BETWEEN THE SINGLE-STRING AND THE THREE VERSIONS OF THE MITIGATED MICROBLAZE DESIGN. (a) OCCUPIED INTERNAL RESOURCES. (b) TIMING PERFORMANCES AND CURRENT CONSUMPTION.

a)

Tested Design	**#FFs**	**#LUTs**	**#LUT-RAM**	**#BRAM**	**#GCLK**	**#IOs**	**#MULT**
Single-string Mblaze design	1,040 (1%)	1,758 (2%)	256	32 (22%)	1 (6%)	78 (9%)	1 (2%)
Mitigated Mblaze design before Replacement of LUT-RAM (Phase 1)	5,596 (8%)	6,242 (9%)	768	96 (66%)	3 (18%)	159 (19%)	3 (6%)
Mitigated Mblaze design after Replacement of LUT-RAM (Phase 2)	13,514 (19%)	31,711 (46%)	0	96 (66%)	3 (18%)	159 (19%)	3 (6%)
Fully Mitigated Design (Phase 3)	13,559 (20%)	32,052 (47%)	0	96 (66%)	3 (18%)	159 (19%)	3 (6%)

b)

Tested Design	**Max. Freq. (MHz)**	**Current Dissipation (A)**
Single-string Mblaze design	77	0.37
Mitigated Mblaze design (phase 1)	66	0.78
Mitigated Mblaze design (phase 2)	66	0.83
Full Mitigated Design (phase 3)	66	0.99

TABLE II
PROTON-INDUCED CROSS SECTIONS OF THE DESIGN 1 AT VARIOUS FLUXES

Flux [p/cm^2/s]	**CLB Upsets / Scrub Cycle**	**Fluence** [p/cm^2]	**Type 1a Error Cross-Section** [cm^2]	**Type 1b Error Cross-Section** [cm^2]	**Type 2a Error Cross-Section** [cm^2]	**Type 2b Error Cross-Section** [cm^2]	**Type 3 Error Cross-Section** [cm^2]
(1) 1.94 x10^7	2 to 7	9.79 x10^{10}	7.56 x 10^{-10}	2.04 x 10^{-11}	6.34 x 10^{-10}	1.43 x 10^{-10}	8.17 x 10^{-11}
(2) 3.87 x10^7	4 to 15	2.49 x10^{10}	8.44 x 10^{-10}	< 4.02 x 10^{-11}	6.03 x 10^{-10}	2.01 x 10^{-10}	1.61 x10^{-10}

TABLE III
PROTON-INDUCED CROSS SECTIONS OF THE DESIGN 2 AT VARIOUS FLUXES

Flux [p/cm^2/s]	**CLB Upsets / Scrub Cycle**	**Fluence** [p/cm^2]	**Type 1a Error Cross-Section** [cm^2]	**Type 1b Error Cross-Section** [cm^2]	**Type 2a Error Cross-Section** [cm^2]	**Type 2b Error Cross-Section** [cm^2]	**Type 3 Error Cross-Section** [cm^2]
(1) 1.70 x10^7	2 to 7	1.00 x10^{11}	7.00x10^{-11}	<1.00x10^{-11}	5.00x10^{-11}	<1.00x10^{-11}	<1.00x10^{-11}
(2) 1.70 x10^8	15 to 30	1.03 x10^{11}	2.92x10^{-10}	9.74x10^{-12}	2.05x10^{-10}	6.82x10^{-11}	<9.70x10^{-12}
(3) 1.70 x10^9	150 to 190	4.86 x10^{10}	1.07x10^{-9}	<2.05x10^{-11}	7.82x10^{-10}	1.65x10^{-10}	3.60x10^{-11}

not being modified by another driver as when implemented in the MicroBlaze design, where the program duty cycle is very high and comparable to 100%. For instance, if the MicroBlaze is running the loop that processes the FFT data, the BRAM scrubber won't be able to interrupt it and scrub the currently executing instruction. Second, the standalone BRAM scrubber design was tested while running at 66 MHz, twice the speed of the same design when inserted in the MicroBlaze design (33 MHz). Therefore the rate of errors (two erroneous bytes located at the same address in the BRAM) should be multiplied by a factor of 4 (22) to be comparable to the runaway reset errors. In addition, as only 7% of the BRAM bits are used for the program code, the mitigated BRAM error cross-sections should also be divided by 14. Therefore, the BRAM error cross sections are adjusted and divided by a factor of 3.5 as shown in column 4 of Table IV.

Table IV shows that at a 1.70×10^8 flux, at least 17% of the runaway resets are due to errors in the BRAM code, while at a 1.74×10^9 flux, 23% of them are caused by code corruption. Some of these code errors cause exceptions. The percentages of runaway resets that are caused by exceptions and their dependence with the flux and the test design are shown in Table V.

Table V shows that an average of 64% of the unrecovered resets has been detected by exceptions in the case of *Design 1* (64% at the flux 1 and 80% at the flux 2). It is clear that the percentage of runaway resets that are detected by exceptions increase with the flux. Indeed, as the number of upsets increase in the same portion of the circuit a bigger code corruption could have occurred. In the case of *Design 2*, exceptions were observed only after an increase of two orders of magnitude of the flux (1.70×10^9) and only 25% of the runaway resets have been detected. At a lower flux (1.70×10^9), although seven resets have been observed, no exceptions have been detected, which shows that not all the illegal states are detected by the exception mechanism. This could be due to the fact that the MicroBlaze was optimized to fit in the Xilinx FPGAs and the exception circuitry has been designed to detect only major illegal operations.

It should be mentioned also that only the unaligned memory access, division by zero and illegal op-code exceptions were observed and for any exception, the one caused by a division by zero is always invoked (single or simultaneously with another exception). Note that all the exceptions occur continuously each time the corrupted code is encountered resulting in a runaway reset until reconfiguration of the DUT.

TABLE IV
PROTON-INDUCED CROSS SECTIONS OF THE BRAM MITIGATION DESIGN AT VARIOUS FLUXES COMPARED TO THE RUNAWAY RESETS ERROR RATES

Flux [p/cm^2/s]	**Cross-Section of BRAM Errors** [cm^2]	**Cross-Section of Runaway Resets** [cm^2]	**Adjusted BRAM Cross-Section** [cm^2]	**Cross-Sections Ratio (BRAM / Runaway Resets)**
(2) 1.70 x10^8	4.22x10^{-11}	6.82x10^{-11}	1.21x10^{-11}	17.68%
(3) 1.74 x10^9	1.33x10^{-10}	1.65x10^{-10}	3.80x10^{-11}	23.03%

TABLE V
PERCENTAGE OF RUNAWAY RESETS CAUSING EXCEPTIONS.

Design	**Flux** [p/cm^2/s]	**Fluence** [p/cm^2]	**Number of Detected Runaway Resets**	**Percentage of the runaway Resets Caused Exceptions**
Design 1	(1) 1.94 x10^7	9.79 x10^{10}	14	64%
(without BRAM scrubber)	(2) 3.87 x10^7	2.49 x10^{10}	5	80%
Design 2	(1) 1.70 x10^7	1.00 x10^{11}	0	/
(with BRAM scrubber)	(2) 1.70 x10^8	1.03 x10^{11}	7	/
	(3) 1.70 x10^9	4.86 x10^{10}	8	25%

From Table V, at least 64% of the unrecoverable reset errors when the BRAM is not scrubbed (*Design 1*) are due to a corruption in the BRAM code as detected by processor exceptions. Because not all exceptions are detected, this is a minimum number. The erroneous instruction is not being corrected because in the case of *Design 1*, the BRAM content is not refreshed, while in the case of *Design 2*, the voters assume that it is a valid instruction. In addition, fewer exceptions have been detected in conjunction with a runaway reset in the case of *Design 2*. It is likely that at the same flux, the code for *Design 2* would be corrupted but not as in the case where the BRAM is not scrubbed. This means that the code might have been corrupted but not till the point where the executed instruction becomes illegal to invoke an exception. This result is valid only at a high flux (when we overwhelm the TMR scheme by having many upsets per scrub cycle), which should not be the case in space applications.

Note the following points that helped to explain the occurrence of the runaway resets that have not been detected by exceptions.

1) Not all the noncoded instructions could be detected by the illegal op-code exception.
2) Modification of an instruction in the code to another one that is still in the MicroBlaze instruction set but enough to modify the execution of the program. For instance, the two instructions bri imm (unconditional branch to an address given by the imm value) and br rb (unconditional branch to an address stored in the register rb) differ only in one bit but both of them are legal. Such a modification is enough to make the program jump to a different part of the code, modify the content of the register rb, miss some of the handshaking between the two FPGAs and therefore cause a runaway reset since the code is not corrected.
3) Each subroutine of the C program code uses at least one or two GPR registers to store the return address. If one bit of the return address is upset, the probability of having the MicroBlaze lose sequence or miss any of the handshaking is high. Note that for optimum fault detection many check points have been inserted in the service FPGA logic so even a jump to the next instruction would desynchronize the two FPGAs.
4) Any code that modifies itself (i.e., dynamically registered interrupt handlers or temporary data storage in mixed code/data sections) could be a source of code corruption. For space applications it would be better to use only static code that could be refreshed by partial reconfiguration (via the configuration scrubber). If resources allow, separate BRAM storage areas should be used for code and data to prevent the possibility of temporary data values overwriting instructions through upsets in the address.
5) Not all illegal operations or states of the MicroBlaze will result in exceptions. This is because the MicroBlaze has been designed to fit in smaller FPGAs and run at a high speed. Indeed, if every illegal state were detected, the exception detection circuitry would occupy a bigger part of the FPGA logic and run slower.

In summary, in any of the studied cases (various particle fluxes), the device error cross section (all type of errors counted) in the case of the fully mitigated design was estimated to be lower than 1×10^{-9}. None of these runaway resets have been detected at fluxes comparable to space fluxes. The obtained results prove successfully the efficacy of the TMR methodology applied in Virtex-II FPGAs to mitigate transient faults likely to occur in space applications.

V. CONCLUSION

In this paper, we have tested a complete solution to mitigate an embedded processor implemented on a Xilinx Virtex II FPGA. This solution is based on continuous external configuration scrubbing, functional-block design triplication, and independent internal BRAM scrubbing (also triplicated). We found that the rate of processor exceptions and resets was kept low and that the use of BRAM scrubbing to protect the code greatly reduced the occurrence of code corruption even at the accelerated fluxes used in beam testing.

The new proposed technique was specifically developed for Virtex FPGAs to cope with transient faults in the user combinational and sequential logic. Its implementation details have been presented as well as the impact on the design performances (device area, frequency of execution and penalty in power consumption). While the low obtained error cross sections prove the high efficacy of this solution, it demonstrates also that the

TMR fault tolerance technique comes with high area and power dissipation penalties.

REFERENCES

[1] F. Lima, C. Carmichael, J. Fabula, R. Padovani, and R. Reis, "A fault injection analysis of virtex FPGA TMR design methodology," presented at the Radiation and Its Effects on Components and Systems, Sep. 2001.

[2] F. Lima(de), S. Rezgui, E. F. Cota, M. Lubaszewski, and R. Velazco, "Designing and testing a radiation hardened 8051-like micro-controller," presented at the Military and Aerospace of Programmable Devices and Technologies Conf., Laurel, MD, Sep. 2000.

[3] G. Swift *et al.*, "Dynamic testing of xilinx virtex-II field programmable gate array's (FPGA's) Input Output Blocks (IOB's)," *IEEE Trans. Nucl. Sci.*, vol. 51, no. 6, pp. 3469–3474, Dec. 2004.

[4] C. Carmichael, B. Bridgford, and J. Moore, "Triple module redundancy scheme for static latch-based FPGAs," presented at the Military and Aerospace of Programmable Devices and Technologies Conf., Laurel, MD, Sep. 2004.

[5] Triple Module Redundancy Design Techniques for Virtex FPGAs, Xilinx Appl. Note XAPP197, C. Carmichael. (2001, Nov.). [Online]. Available: http://www.xilinx.com/bvdocs/appnotes/xapp197.pdf

[6] MicroBlaze Processor Reference User Guide, Xilinx, Inc., Aug. 2004. Embedded Development Kit (EDK 6.3), UG081, Version 4.0.

[7] FFT C Code, T. Roberts and M. Slaney. (1994, Dec.). [Online]. Available: http://www.jjj.de/fft/int_fft.c

[8] TMR Tool User Guide, Xilinx, Inc., UG156, Version 6.2.3 (2004, Sep.). [Online]. Available: http://support.xilinx.com/products/milaero/ug156.pdf

[9] *Triple Module Redundancy Design Techniques for Virtex FPGAs*, Nov. 2001. Xilinx Appl. Note 197.