2010 International Conference on Reconfigurable Computing

Fault Injection Results of Linux Operating on an FPGA Embedded Platform

Joshua S. Monson #1, Mike Wirthlin #2, Brad Hutchings #3

# *Department of Electrical and Computer Engineering, Brigham Young University*
459 Clyde Building, Provo UT, 84602, United States Of America
1 jsmonson@gmail.com
2 wirthlin@ee.byu.edu
3 brad_hutchings@byu.edu

***Abstract*—An FPGA-based Linux test-bed was constructed for the purpose of measuring its sensitivity to single-event upsets. The test-bed consists of two ML410 Xilinx development boards connected using a 124-pin custom connector board. The Design Under Test (DUT) consists of the "hard core" PowerPC, running the Linux OS and several peripherals implemented in "soft" (programmable) logic. Faults were injected via the Internal Configuration Access Port (ICAP). The experiments performed here demonstrate that the Linux-based system was sensitive to 92,542 upsets-less than 0.7 percent of all tested bits. Each sensitive bit in the bit-stream is mapped to the resource and user-module to which it configures. A density metric for comparing the reliability of modules within the system is presented.**

I. Introduction

Over the last decade several projects have illustrated the interest of using Linux Operating Systems (OS) on space-based computing platforms. NASA, for example, sponsored the FlightLinux project which culminated by demonstrating a reliable Linux OS on the UoSat12 satellite [1]. Compatibility with ground systems is only one of the many reasons to use Linux on space-based computing platforms. Linux also provides access to well used libraries and free development tools.

Reliable hardware is essential for Linux to operate; however, integrated circuits aboard space-based computing platforms are susceptible to failures known as Single Event Upsets (SEUs). SEUs are random bit flips caused by radioactive particles that collide with the integrated circuits (ICs). Typically ICs are specially designed or hardened to operate correctly in the presence of radiation. Unfortunately, radiation-hardened parts are expensive and usually two or three silicon generations behind the state of the art [2].

Field Programmable Gate Arrays (FPGAs) are among the state of the art components that are of interest in space-based computing. FPGAs are microchips that contain an array of logic and interconnect that can be programmed and reprogrammed to perform almost any digital function. Designing for FPGAs is faster and less expensive than designing for Application Specific Integrated Circuits (ASICs). Additionally, reprogrammability allows designers to remotely fix bugs that appear after the platform has launched. These features make FPGAs ideal for space-based platforms [3] [4] [5].

SRAM-based FPGAs store their programming bits in memory cells on the device. Flipping the bits contained in these memory cells may modify the behavior of a logic circuit on an FPGA (until it is reconfigured). For example, an SEU has the potential to change the contents of a Look-Up Table (LUT), connect nets together, or to completely disconnect a net.

Mitigation techniques are used on FPGAs to prevent SEUs from affecting the output of a design. Bit-stream scrubbing [6], Triple Modular Redundancy [7] (TMR), and Partial TMR (PTMR) [8] are common methods of mitigating the effects of SEUs. TMR uses redundant circuits and majority voters to improve the reliability of FPGA designs. PTMR is a reduced form of TMR that has been shown (on small circuits, at least) to reduce area overhead with only a small reduction in reliability.

In FPGA-Embedded Linux systems using "hard core" processors, important peripherals (such as the memory controller) are implemented in the reconfigurable fabric of the FPGA and are susceptible to SEUs. SEU induced failures in these components have the potential of crashing the kernel. Understanding each peripheral's likelihood of causing a kernel failure due to a SEU aids in understanding the reliability of the system and in creating a more reliable system at the lowest cost. To gain this understanding, we constructed a test-bed that allows us to simulate SEUs in the FPGA fabric surrounding a "hard core" embedded processor running a Linux kernel using a process known as fault injection. This is the first work that we are aware of that analyzes failures according to FPGA circuit categories (logic, routing) and user modules (memory interface, processor interface, etc.).

II. Previous Work

The development of fast, comprehensive fault injection systems has been the focus of much of the previous work [9] [10] [11]. These fault injection systems are able to emulate upsets in FPGA fabric and microprocessor cache lines and special registers. Rather than performing exhaustive tests, these fault injection systems use a probabilistic model to statistically determine the reliability of a design.

The work presented by Sterpone and Violante [12] performed fault injection experiments on the memory image of a

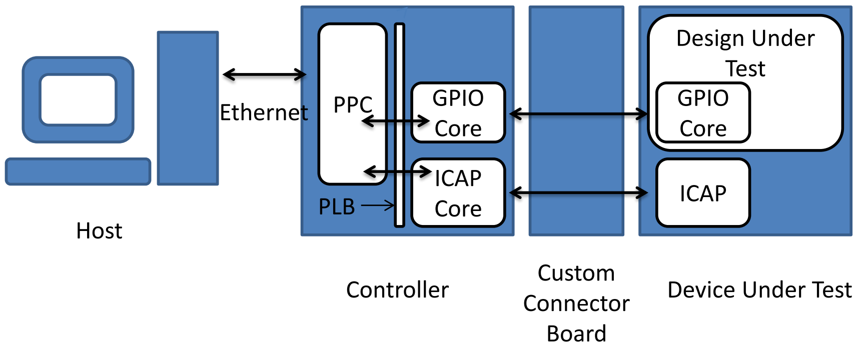


Fig. 1. A block diagram of our fault injection system.

Linux micro kernel running on a Xilinx Microblaze processor implemented in soft-logic. Their work focused on memory faults during the bootstrapping process but did not examine the sensitivity of the circuitry/logic that implements their system. Essentially, their fault-injection process modified the content of the memories that contain the Linux program. In contrast, our effort injects faults directly into the hardware implementation and analyzes the sensitivity of a full Linux kernal system (rather than a micro kernel) to circuit and logic failures that may be caused by SEUs.

In another work Johnson et. al.[13] describe the validation of a fault injection simulator using a proton accelerator. Their analysis used the bit-stream offsets of sensitive bits to compute the row and column locations of the sensitive bits. Doing this allowed them to create a "map" of the FPGA showing the locations of sensitive bits. Their fault injection simulator was able to predict the locations of an impressive 97% of the upsets caused by the proton accelerator. In this work we also endeavor to improve the verification of fault injection experiments. For our fault injection experiment we studied the bit-stream to determine the relationships between configuration bits and FPGA resources; this allowed us to take the next logical step and map sensitive bits to FPGA resources, and further, map resources to user-design modules.

III. System Architecture

Our fault injection system consists of a Linux-based host PC and two Xilinx ML410 development boards. The ML410 is a Virtex-4 (FX60)-based development board that contains two DDR memories, an Ethernet PHY and connectors, serial ports, a SystemAce controller, and other components not utilized in this experiment [14]. One ML410 acts as the experiment controller and the other acts as the Device Under Test (DUT). The PPCs on both the DUT and the experiment controller are running the Linux 2.6.29 kernel provided by Xilinx (for more information on the kernel see http://xilinx.wikidot.com). Figure 1 is a block diagram showing how these components were connected. The Linux-based host is connected to the controller via Ethernet and RS-232. The two ML410 boards are connected using a 124-pin custom connector board.

The Linux-based host provides a simplified interface to the embedded Linux Kernel running on the experiment controller (ML410 Board). Experiment software was developed on the host and cross-compiled for the PowerPC 405. The host maintained a NFS mounted file system that provided a simple means for transferring software, test vectors, and results to and from the experiment controller. The RS-232 connection acted as the console for the Linux kernel.

The experiment controller injects faults using Partial Run-time Reconfiguration (PRR) through the DUT's Internal Configuration Access Port (ICAP). Virtex-4 FPGAs support PRR via the ICAP and allow the designer to read and write the configuration memory at 400 MB/sec. All DUT test designs communicate directly with the experiment controller via routing (contained in the DUT test design) that connects the DUT's ICAP to the custom connector board. The experiment controller injects faults through this connection. In cases where an injected fault causes the DUT's ICAP interface to fail, the controller can completely reconfigure the DUT using the external configuration interface. The control and communication circuitry for the ICAP is implemented on the experiment controller board to maximize available circuitry in the DUT. The ICAP interface on the experimental-controller board is controlled and configured by software and supports a wide variety of experiments.

The testbed facilitates novel methods of control and test vector read back. For example, one of our experiments (not the fault injection experiment discussed later in the paper) required that the clocks be initially disabled and then re-enabled before the experiment was performed. To implement this, LUTs with constant outputs of '1' were wired to the resets of the digital clock managers (DCMs). After the fault was injected, the LUTs were reconfigured such that their outputs were constant '0's, this activated the DCMs and started the DUT. This method provided control without significant modifications to the DUT.

Another experiment produced a 256-bit result vector that

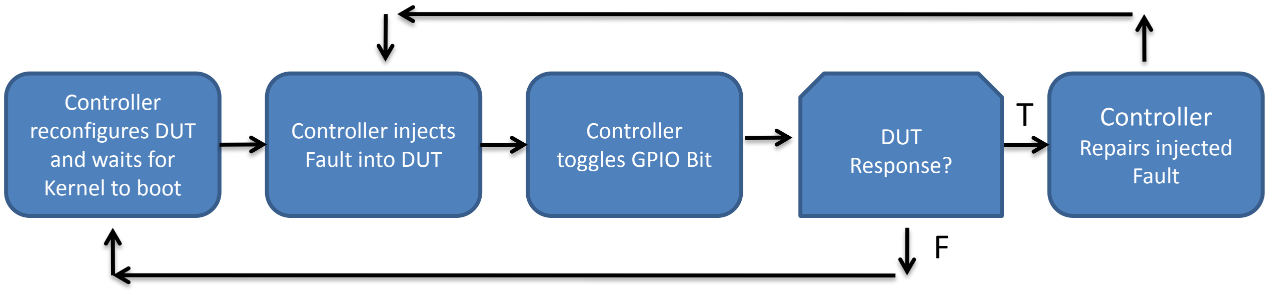

Controller reconfigures DUT and waits for Kernel to boot
Controller injects Fault into DUT
Controller toggles GPIO Bit
DUT Response?
T
F
Controller Repairs injected Fault

Fig. 2. Flowchart of the fault injection routine.

was read at the end of the experiment. By parsing the logic allocation file we were able to identify where the result vector was stored in the bit-stream. Rather than having to create a state machine to send it back to the controller one 32-bit word at a time we were able to capture the result and read it back through the ICAP.

IV. Description of Fault Injection Experiment

Fault injection is the process of emulating SEUs by flipping bits in the bit-stream during run time. First, a test bit is selected. The ICAP is used to read back the frame containing the test bit. The test bit is then inverted and the frame is written back into the bit-stream through the ICAP. If the fault causes the DUT to fail, the test bit is considered “sensitive”, otherwise the test bit is considered “not sensitive.”

The experimental procedure is illustrated in Figure 2. The controller reconfigures the DUT and waits for the Linux kernel to boot. After the kernel has booted, the controller then injects a fault into the DUT. The controller determines if the Linux kernel is still active by toggling its GPIO bit and waiting for a response from the DUT. If the Linux Kernel on the DUT has failed, no response will be received and the bit will be considered sensitive. If the DUT responds, then its Linux Kernel is still active and the corresponding bit is considered not sensitive. Finally, the controller resets the configuration bit to its original value to prepare for the next iteration of the test.

Although fault injection is a proven and very effective way to emulate SEUs, there are a few areas where it cannot achieve complete coverage. For example, reading and writing the block rams (BRAMs) via the ICAP may cause side effects that artificially increase the bit-sensitivity count. SRL16s and LUT RAMs are corrupted during frame reads and writes (setting the GLUTMASK_B to a ’0’ prevents this). IOB blocks are avoided as well; these bits control FPGA I/0 pins and arbitrarily flipping these bits may cause board or device damage. All the bits corresponding to interconnect and logic were tested except for the bits in the IOB interconnect and control bits and bits that were set in the mask file.

Accessing the ICAP on the DUT requires that signals be routed through the reconfigurable fabric of the DUT. These signals are vulnerable to fault injection testing. A failure in the ICAP circuitry prevents faults from being injected and repaired. After each fault was injected the ICAP was tested to ensure that it was still working properly. If the ICAP circuitry had been damaged the FPGA was completely reconfigured.

TABLE I
Device Logic Utilization

Resource	Used	Available	Utilization
Slice Registers	6,271	50,560	12%
4-input LUT	5,688	50,560	11%
Block RAMs	53	232	23%
SRL16 Shift Registers	281	50,560	.5%
LUT RAMs	0	50,560	0%

V. Analysis of Results

Our results will be presented using three different analyses. The general overview analysis analyzes the overall number of sensitive bits and describes the reliability of our unmitigated Linux System. In the FPGA resource analysis, the sensitive bits analyzed in the general overview are mapped to the resources they control. In the user circuit analysis, the sensitive resources are mapped to the module which they belong (memory controller, UART, etc.) and a metric for comparing the sensitivity of modules is explained.

These three analyses are interesting because they provide a designer with data that answers questions such as: what is the general sensitivity of the DUT? what resources cause most of the design sensitivities? which modules in the DUT are most sensitive to SEUs? Answers to these questions could aid in the directed application of mitigation techniques and possibly save design area or suggest FPGA architecture improvements.

In addition, these analyses allow those performing fault inject experiments to verify their results by confirming that sensitive bits actually have the ability to effect the design. This is done by mapping sensitive bits to actual resources in the FPGA and verifying that they belong to or can effect a component in the DUT.

A. General Overview Analysis

Table I shows the device utilization of the DUT. While 23% of the BRAMs are instantiated less than half of them are actually used by the Linux kernel, making their effective

TABLE II
SUMMARY OF BITS TESTED/NOT TESTED

Bits Tested	13,757,308
Bits Not Tested	7,140,228
BRAM Content (excluded)	5,373,952
Mask File Bits (excluded)	821,636
IOB Bits (excluded)	944,640
Total Bits	20,960,512
Bits Used in DUT (approx.)	2,061,914

TABLE III
SUMMARY OF SENSITIVITIES

	Tested Sensitive Bits	SRL16 Bits	BRAM Content	Total
DUT	29,167	4,496	58,880	92,543
ICAP	1,357	0	0	1,357

utilization about 10%. The BRAMs not used by the Linux kernel are inserted by Xilinx's Embedded Development Kit (EDK) and are used to boot the PowerPC. Overall, the device utilization is between 10% and 12%.

Table II gives a summary of the bits that were and were not tested. As the table shows there were 13,757,308 bits tested out of the 20,960,512 in a normal bit-stream for this device. Although Table II shows that 7,140,228 in the FPGA were not tested, the sensitivity of the BRAMs and SRL16 Bits were estimated as discussed in the next paragraph. We did not attempt to estimate the majority of the masked bits or the IOB bits.

Table III presents the number of sensitivities found in the DUT and ICAP circuitry. The SRL16 bits were estimated by assuming that all content bits in the shift register would be sensitive. The same assumption was made for all BRAMs used by the Linux kernel (some were instantiated by EDK but not used by the kernel). The IOB bits were not tested and no attempt was made to estimate the effect of their sensitivity on the device.

The Mean Time Between Failure (MTBF) of our unmitigated DUT is 26.4 days. The MTBF was calculated using Equation 1. This result includes both the reconfigurable fabric and the embedded PowerPC processor. The failure rate of an individual configuration bit within the Virtex 4 architecture in the geostationary earth orbit is 2.78×10^{-7} [15] upsets per day. The sensitivity of the PowerPC processor was derived from the results of [10]. Using their results we calculated that the PowerPC contains 43,681 sensitive bits. Adding this number to our total number of sensitive bits produces a total of 136,224 sensitive bits for the whole system.

$$MTBF = \frac{1}{\lambda} = \frac{1}{2.78 \times 10^{-7} \times 136,242} = 26.4 \; days. \quad (1)$$

With a MTBF of 26.4 days, the Linux kernel will spend 30-60 seconds per month rebooting. If this is not sufficiently reliable, mitigation may be required.

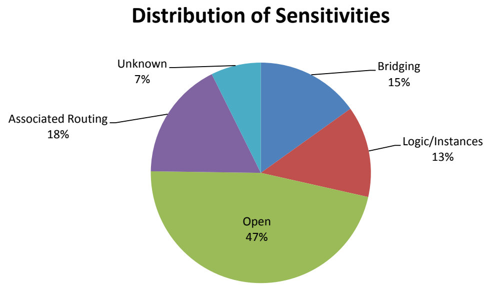


Fig. 3. The distribution of sensitivities among utilized resources.

B. FPGA Resource Analysis

A FPGA Resource Analysis provides insight into which resources caused the bit sensitivities in the FPGA-embedded Linux design. This type of analysis is intended to provide insight into why the DUT failed. This is done by mapping the sensitive bits analyzed in the previous section to the resources that they program. Understanding the resources involved in causing a design to fail could suggest resource-level mitigation approaches or future improvements to FPGA architectures.

Failures were placed into three sensitivity categories: Logic, Routing and Unknown. Routing sensitivities were further categorized into open routing sensitivities, bridging routing sensitivities, and associated routing sensitivities. Figure 3 shows the distribution of the different types of sensitivities identified among the sensitive bits. Unknown sensitivities were those sensitivities bits that could not be mapped to a utilized resource in the DUT.

Logic Sensitivities consisted of all sensitivities that were associated with the attributes of a Logic Slice. As Figure 3 shows, the number of sensitivities in the logic of the FPGA is relatively low when compared with that of the routing.

Routing Sensitivities accounted for 80% of the all the sensitivities in the design. The routing sensitivities are broken down into three categories: open, bridging, and associated routing failures. The open routing sensitivities were all the routing sensitivities that would completely disconnect a net used in the design. Bridging sensitivities include all those sensitivities that had the effect of connecting two nets together. The associated routing sensitivities included all sensitive bits that could be loosely associated with the net, but did not disconnect the net or connect it with to other nets. It is possible that these "loosely-associated" bits drive a wire that is used by another net, or that the change in the routing caused the net to no longer meet timing.

Unknown Sensitivities account for 7% of the design sensitivities. Only 4.1% of these bits were sensitive when tested a second time. While the reason they tested positive in the original test is unaccounted for, it is possible that some of these bits became scapegoats for previously tested bits whose effects had remained dormant until well after their test was over. Another possibility is that these bits, during the second test, were not tested during its sensitive context. A final possibility

TABLE IV
SENSITIVE BITS TO RESOURCE USAGE

Module	Sens. bits/Resource
DCM	7.483
GPIO	3.930
ICAP	3.264
PLB Bus	3.178
PPC	2.642
System Reset	2.365
Memory Controller	1.554
UART	1.460
IIC Bus	.813
Linux System	1.818

is that there may still be some undiscovered side effects of dynamic partial reconfiguration in this Virtex 4 device that artificially caused these sensitivities in the design.

These results demonstrate that identifying the cause of design failure is not completely out of reach. In our unmitigated design routing failures were the dominant sensitivities. This may not be the case after applying mitigation techniques. Identifying the failed resources in mitigated designs could help to identify unknown weaknesses in current mitigation techniques.

C. User-Circuit Analysis

A User-Circuit Analysis analyzes the reliability of a design on a module by module basis. In this analysis, the sensitive resources identified in the FPGA Resource Analysis are mapped to the cores in which they belong. This information can be used to make decisions on the amount of redundancy used to increase the reliability of a system on a module by module basis.

The charts in Figures 4(a), 4(b), and 4(c) demonstrate that in this system resource usage is not directly proportional to the sensitivity of a module. Figures 4(a) and 4(b) give a breakdown of nets and logic functions (LUTS, FF, etc) in the modules that comprise the DUT. Figure 4(c) is a chart showing the percentage of sensitivities in each module of the DUT. Notice that the memory controller occupies 70% of the nets and instances (of FPGA resources) in the design yet is responsible for only 59% of the bit-sensitivities. On the other hand, modules such as the processor local bus (PLB) and PPC have a higher percentage of sensitivities compared to the number of resources they occupy.

The disparity between a module's resource usage and its number of sensitive bits can be explained by the fact that this is a microprocessor system. The criticality and frequency of usage impact the sensitivity of a module. For example, the DCM, occupies 1% of the design nets and instances, but accounts for 3% of the design sensitivities. This is likely because the clock generated by the DCM is always critical to the correct operation of the system. This is also seen in the senstivities of the PLB bus and the PPC (the PPC connects the PowerPC-405 to the PLB Bus). These likely have a higher density of sensitivities than the Memory Controller because they must be used for every transaction across the PLB bus.

To facilitate the direct comparison of the sensitivity of multiple modules we introduce a sensitive-bit density metric measured in sensitive bits per unmitigated resource. This metric is calculated by dividing the number of sensitive bits in a module by the number of unmitigated resources in the module (instances + nets). This metric allows a designer to directly compare the sensitivity of two modules within the same system. The use of unmitigated resources is suggested to maintain linearity and avoid exaggerating the increase in reliability when applying redundancy techniques.

Table IV presents the sensitive bit-density for each module in the DUT and the entire Linux System. The sensitive-bit density metric can be used to create a priority queue to determine which modules should receive full, partial or no TMR. The idea being that differing levels of TMR can be used to achieve similar sensitive bit-density levels.

VI. FUTURE WORK

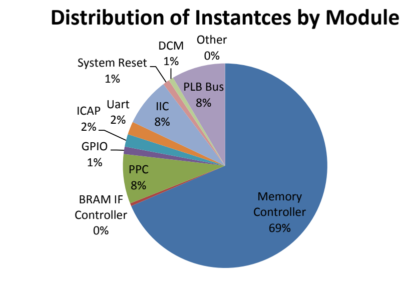
The sensitive bit-density metric is measured in sensitive bits per resource. This provides a good estimate of the sensitivity of a module. Additionally, the total numbers of nets and instances for each module is easy to obtain from the Xilinx Description Language (XDL) file which can be generated by the Xilinx tools. A better unit for the sensitive bit-density metric would be sensitive bits per configuration bit. This would be more accurate due to the fact that different resources are configured by different numbers of configuration bits. For example, a long net uses a different number of configuration bits than a short one, which may use a different number of configuration bits than a LUT. Therefore, considering a net as a single resource is not as accurate as considering the number of configuration bits required to program the resource.

To calculate the sensitive bit-density metric using sensitive bits per configuration bit, work would have to be done to determine the exact number of bits used to configure each resource. Further, a study could be done to determine the number of sensitive bits in different use cases. For example, lets say a given resource is configured by 15 configuration bits, however, under a given use case only 10 bits actually change the output of the resource. This resource may be a LUT or a DSP or a FF. Resources such as routing may even have more sensitive bits than configuration bits.

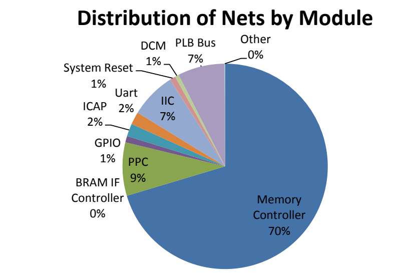
Further work could be done to improve the speed at which Linux Reliability tests are carried out. Due to the time-penalty introduced by rebooting the Linux kernel for each sensitivity discovered the time of our full fault injection test was 8 days. Work could also be done to identify failure modes and recovery mechanisms for the Linux kernel.

VII. CONCLUSION

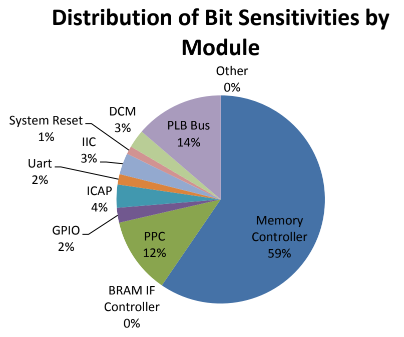
A reliable Linux OS can be useful tool on an FPGA embedded System. Fault injection testing is an important first step in testing the reliability of FPGA Embedded Linux Systems. Our test-bed provides an ideal platform for fault injection and



(a) The distribution of instances among modules in the DUT.



(b) The distribution of nets among modules in the DUT.



(c) The distribution of sensitivities among modules in the DUT.

Fig. 4. Pie charts showing user-circuit level statistics.

other useful experiments investigating the low-level details of the FPGA. Using different analyses in fault injection may help in identifying the lowest cost SEU mitigation techniques for FPGA-Embedded Linux Systems and provide additional confidence in fault injection results.

Our sensitive bit-density metric provides a means of directly comparing the sensitivity of modules within a design. This leads to directed application of redundancy techniques such a TMR and Partial TMR that may save precious design resources.

Acknowledgment

We would like to express our thanks to the Information Sciences Institute-East (ISI) for providing the ML410 boards and custom connector board. We would especially like to thank Neil Steiner for his assistance in the initial stages of our project.

References

[1] B. Ramesh, T. Bretschneider, and I. Mcloughlin, "Embedded linux platform for a fault tolerant space based parallel computer," *In Proceedings of the Real-Time Linux Workshop*, pp. 39–46, 2004.
[2] D. S. Katz. (2004) Application-based fault tolerance for spaceborne applications. [Online]. Available: http://hdl.handle.net/2014/10574
[3] M. Caffrey, "A space-based reconfigurable radio," in *Proceedings of the International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA)*, T. P. Plaks and P. M. Athanas, Eds. CSREA Press, June 2002, pp. 49–53.
[4] M. Caffrey, K. Morgan, D. Roussel-Dupre, S. Robinson, A. Nelson, A. Salazar, M. Wirthlin, W. Howes, and D. Richins, "On-orbit flight results from the reconfigurable Cibola Flight Experiment satellite (CFESat)," in *Proceedings of the 17th IEEE Symposium on Field Programmable Custom Computing Machines (FCCM '09)*, April 2009.
[5] M. Caffrey, K. Katko, A. Nelson, J. Palmer, S. Robinson, D. Roussel-Dupre, A. Salazar, M. Wirthlin, W. Howes, and D. Richins, "The Cibola Flight Experiment," in *Proceedings of the 23rd Annual Small Satellite Conference*, August 2009.
[6] C. CARMICHAEL, M. CAFFREY, and A. SALAZAR, "Correcting single-event upsets through virtex partial configuration," *Xilinx Application Notes*, vol. 1.0, 2000.
[7] F. L. Kastensmidt, L. Sterpone, L. Carro, and M. S. Reorda, "On the optimal design of triple modular redundancy logic for sram-based fpgas," in *DATE '05: Proceedings of the conference on Design, Automation and Test in Europe*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 1290–1295.
[8] S. Baloch, T. Arslan, and A. Stoica, "Probability based partial triple modular redundancy technique for reconfigurable architectures," in *Aerospace Conference, 2006 IEEE*, 2006, p. 7 pp. [Online]. Available: 10.1109/AERO.2006.1656007
[9] U. Legat, A. Biasizzo, and F. Novak, "Automated SEU fault emulation using partial FPGA reconfiguration," in *Design and Diagnostics of Electronic Circuits and Systems (DDECS), 2010 IEEE 13th International Symposium on*, 2010, pp. 24–27. [Online]. Available: 10.1109/DDECS.2010.5491825
[10] M. S. Reorda, L. Sterpone, M. Violante, M. Portela-Garcia, C. Lopez-Ongil, and L. Entrena, "Fault injection-based reliability evaluation of SoPCs," in *Test Symposium, 2006. ETS '06. Eleventh IEEE European*, 2006, pp. 75–82. [Online]. Available: 10.1109/ETS.2006.24
[11] P. Civera, L. Macchiarulo, M. Rebaudengo, M. Reorda, and M. Violante, "FPGA-based fault injection for microprocessor systems," in *Test Symposium, 2001. Proceedings. 10th Asian*, 2001, pp. 304–309. [Online]. Available: 10.1109/ATS.2001.990301
[12] L. Sterpone and M. Violante, "An analysis of SEU effects in embedded operating systems for Real-Time applications," in *Industrial Electronics, 2007. ISIE 2007. IEEE International Symposium on*, 2007, pp. 3345–3349. [Online]. Available: 10.1109/ISIE.2007.4375152
[13] E. Johnson, M. Caffrey, P. Graham, N. Rollins, and M. Wirthlin, "Accelerator validation of an FPGA SEU simulator," *Nuclear Science, IEEE Transactions on*, vol. 50, no. 6, pp. 2147–2157, 2003. [Online]. Available: 10.1109/TNS.2003.821791
[14] *ML410 Embedded Development Platform User Guide*, San Jose, CA, USA, 2008.
[15] G. Allen, "Virtex-4VQ dynamic and mitigated single event upset characterization summary," Jan. 2009. [Online]. Available: http://hdl.handle.net/2014/41104