Error Correction Algorithm for SRAM Based FPGA

A. T. Salawudeen[1], E. O. Haruna[2 *], H. Salawu[3], S. M. Yusuf[1] and S. Muhammad[1]

[1] Department of Computer Engineering, Ahmadu Bello University, Zaria
[2] Centre for Satellite Tech. Devt, National Space Research and Development Agency Abuja, Nigeria.
[3] Physics Department, Federal University, Lokoja.
[4] Department of Communication Engineering, Ahmadu Bello University, Zaria
*ocholiharuna@gmail.com

Abstract— *This paper presents an improved Frame Level Redundancy Scrubbing (FLR) algorithm that uses Cyclic Redundancy Check (CRC) as an error detection technique for configuration memory scrubbing, is developed as a solution to mitigate Single Event Upset (SEU) through upset detection and correction. Fault injection was performed on FPGA configuration memory frames on a different number of modules to emulate SEU. The improved FLR algorithm was implemented and system level simulation was carried out using MATLAB. The performance of the improved FLR algorithm was compared with that of the existing FLR algorithm using error correction time and energy consumption as metrics. The results of this work showed that the improved FLR algorithm produced 31.6% improvement in error correction time and 61.1% improvement in energy consumption over the existing FLR algorithm.*

Keyword—FPGA, SRAM, Scrubbing, FLR, SEU, configuration memory, a logic bit(s).

INTRODUCTION

SRAM FPGAs are complementary metal oxide semiconductor (CMOS) devices with a special characteristic of reconfigurability making them desirable for use in systems with evolving technology [1]. The use of FPGAs has been shown to provide high computational density and efficiency for many computing applications by allowing circuits to be customized to any application of interest. They are attractive to critical applications due to their high performance, power consumption, and reconfiguration capability [2], and can be re-configured in the field, design updates can be performed while the device is still operational. Compared to application specific integrated circuits (ASICs), whose functions cannot be altered after fabrication, SRAM-based FPGAs have the advantage of being reprogrammed and providing a lower cost per device in small quantities, therefore, there is great interest in exploiting these benefits in space and other radiation environments [3]. Mapping refers to the configuration of the FPGA device [4]. In SRAM based FPGAs, the mapped circuit is totally controlled by the configuration memory which is composed of SRAM cells [5]. A modern generation FPGA have tens of thousands to millions of system gates, with hundreds of millions of configurations bits, dominating the SRAM cells in the device [6].

While SRAM-based FPGAs offer several advantages for critical based operations, they are sensitive to SEUs. Thus, when a fault changes the state of an SRAM cell, this event is referred as SEU [7]. In other words, SRAM-based FPGAs are more prone to soft errors since a radiation strike in a configuration memory has a permanent effect on the functionality of the mapped design [8]. The SRAM-based FPGAs are especially sensitive to SEUs within the configuration memory of the device. The configuration memory defines the operation of the FPGA resources and upsets in the configuration memory can change the operation of the circuit. To ensure proper operation SRAM-based FPGA circuit designs must mitigate against any configuration memory SEU which could alter the design [9].

The configuration memory of SRAM-based FPGAs is arranged into segments called "configuration frames", and this represents the largest portion of the memory cells in the device. Some factors that increase the susceptibility to soft errors are the reduction of the transistor size and the lower voltage operations of these SRAM memory cells [1].

Technology scaling leads to an increase in memory density as well as the probability of SEUs and MBUs in adjacent bits due to particle strike. Soft errors (reversible errors) can be generally tolerated in consumer electronics but can have adverse effects in mission-critical applications using SRAM-based FPGA [10]. Soft errors in the configuration memory bits of SRAM based FPGAs have a persistent effect and they remain until the original configuration is rewritten [1].

The presence of high energy protons, heavy ions, and galactic cosmic rays in the space and other radiation environment cause a number of problems for electronics, including FPGAs. This radiation can induce a number of negative effects including upsets in the internal state of the device and can cause several problems in FPGA-based systems. As mentioned earlier, SEUs can corrupt the configuration memory of the device causing the design configured on the device to operate incorrectly [11]. Configuration memory scrubbing is a technique used to correct or mitigate against errors (SEUs) in SRAM-based FPGA after they are detected by other techniques such as CRC. [12]. SEU mitigation is crucial for systems operating in harsh environments with high levels of cosmic radiation. Energetic particles generate charge as they traverse the semiconducting materials which get deposited inducing voltage transients to the interconnected nodes. [10].

METHODS AND MATERIAL

In this section, the relevant information needed to implement the proposed algorithms are provided.

2.1 Single Event Upset

SEU is a form of Single Event Effects (SEE) which are a

change of logic states or transients in a device induced by energetic radiation particles from the environment in which the device is operated. A single event upset is the change in state of a digital memory element caused by high energy particles such as protons, neutrons, or heavy ions. If the ionizing particle passes from one node to another, and the charge is greater than the device specific critical charge, (critical charge is defined as the minimum amount of charge to flip the data stored in a memory element [18]), this charge transfer can change the voltage level of critical nodes within the configuration memory cell of an FPGA such that the improved voltage level reflects the opposite state of the cell (that is changing a logic ''1'' to a logic ''0'' or a logic ''0'' to a logic ''1''). The feedback nature of static latches will preserve this new value and the original value will be lost [3].

A single bit flip can have significant consequences on FPGA functionality and a serious impact on the design itself. For example, a single bit flip in a flip-flop in the Configurable CLB or Lookup Table (LUT) can change a Boolean AND function to a different Boolean function, in other words, any bit-flip in the LUT may cause the logic implemented by it to produce a faulty output as long as it is not corrected. A single bit flip can also change the connections in the FPGA's routing network. The results of an SEU in an FPGA's configuration memory can be unpredictable [19]. Figure 1 demonstrates what may happen to the two-input ''AND'' gate. When upsets occur in the configuration memory, the first configuration upset is a change in the routing configuration data as shown in Figure 2 that disconnects one input from the ''AND'' gate. The second configuration upset as depicted in Figure 3 is a change in the look-up table content of the ''AND'' gate and modifies the operation logic function (it no longer performs the ''AND'' function rather it now performs an ''exclusive OR'' function). In both cases, upsets in the configuration memory change the behavior of the circuit so that the circuit no longer performs the function intended by the circuit designer.

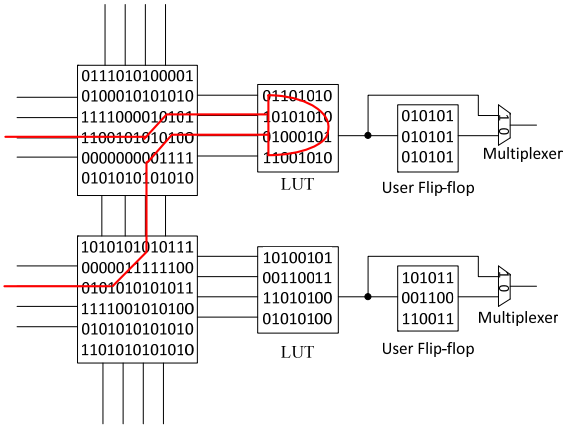

Figure 1: Configuration Memory Used to Specify Logic and Routing [19]

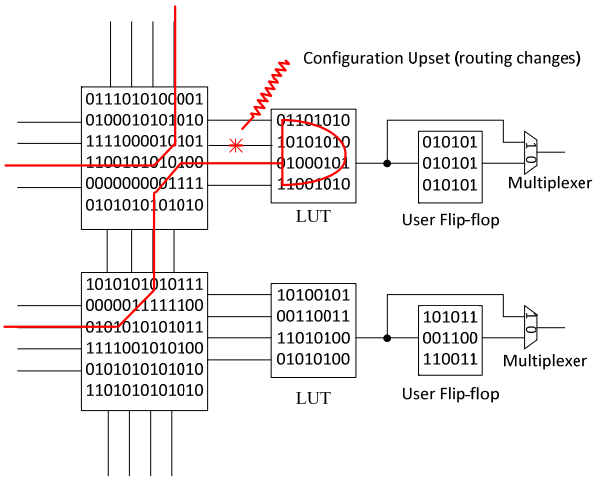

Figure 2: Upset in Routing [19]

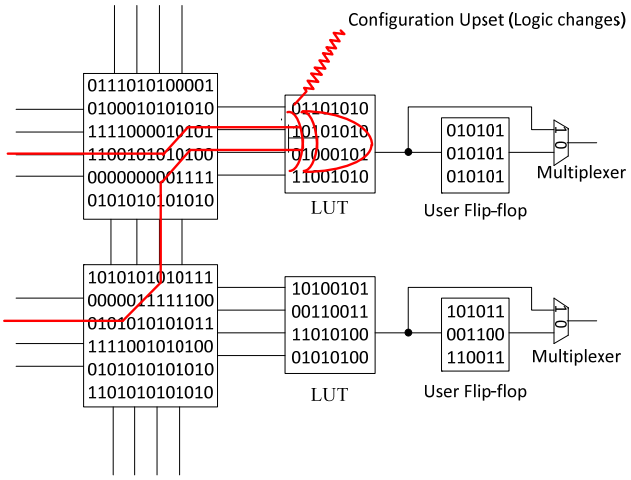

Figure 3: Upset in Logic [19].

2.2 Cyclic Redundancy Check

Besides transferring data as quickly as possible, storage systems have to maintain data integrity, assuring correctness of storage data. Algorithms for data integrity becomes an important component of such system. CRC as an error detection mechanism that maintains data integrity and can be used during readback process on each frame header storing only the check word rather than the entire frame of the configuration data [20, 21].

In the encoding process of an r-bit CRC, after selecting a fixed generator polynomial G(x) having a degree r and M(x) is the message word or data in the configuration memory. Therefore, a multinomial is generated having k-bits of the message word with an appended r-bits redundancy. The following steps are executed (Zhang & Ding, 2011):

A. Generating a multinomial by multiplying X^{n-k} with $M(x)$ to give

$$x^{n-k}M(x) \quad (1)$$

B. Dividing $x^{n-k}M(x)$ by $G(x)$ results in a quotient of $Q(x)$ and a remainder of $R(x)$. The degree of $R(x)$ must be smaller than the degree of $G(x)$, that is r.

The result of the division yields:

$$\frac{X^{n-k}M(x)}{G(x)} = Q(x) + \frac{R(x)}{G(x)} \quad (2)$$

where:

$x^{n-k}M(x)$ is the encoded message word

$R(x)$ is the remainder which is the CRC value.

$Q(x)$ is the quotient

Therefore, the encoded message word can be expressed as:

$$X^{n-k}M(x) = G(x)Q(x) + R(x) \quad (3)$$

The data in the storage device is the dividend and the remainder become the result. Every bit in the storage device requires one exclusive-OR (XOR) and one shift operation to the left by the degree of a polynomial minus one bit [22].

2.3 Power Consumption

Configuration memory scrubbing comes with power consumed by the scrubber circuitry. This is because power overhead is driven by the scrub or readback rate. Total power consumption is composed of static and dynamic power. Static power is related to the transistor leakage current and dynamic power is related to the switching activity of transistors and its value depends on the rate of switching. The static power consumption can be considered negligible and the dynamic power is the main contributor to the total power consumption.

The total power consumption (P_T) is the sum of the dynamic power (P_D) and Static Power (P_S). The total power, static and dynamic power is given equation 4, 5 and 6 respectively [23]:

$$P_T = P_D + P_S \quad (4)$$

$$P_S = V_{CC} + I_{CC} \quad (5)$$

Where;

V_{CC} is the voltage level and I_{CC} is the leakage current

$$P_D = \sum_{i=1}^{n} \beta_i C_i f V_{CC}^2 \quad (6)$$

Where;

n = number of toggling nodes, β_i = switching activity, C_i = load capacitance of the node, f = clock frequency and V_{cc} = transistor source voltage

Since all the transistors in an SRAM of an FPGA are turned on independently to the design synthesized into the configurable memory, it is expected that the static power of a design is almost constant when compared to the total power consumed of the device. In order to estimate the power overhead of a TMR system implemented in an SRAM-based FPGA, it is assumed that the use of three modules will mainly impact the dynamic power component [23].

1 DEVELOPED SCHEMES

The Methodology adopted in this are as follows:

A. Replication of the FLR scrubbing algorithm which requires the following steps to be carried out:
   1. Generate configuration memory in the MATLAB environment.
   2. Assign Logic bits to the frame cells representing the mapped designed.
   3. Triplicate the configuration memory to form three modules.
   4. Perform fault injection campaign on the three configuration memory module to randomly flip bits in the frame cells.
   5. Compare logic bits in the same cell position with the same frame address in the three configuration memory module for SEU detection.
   6. Perform bit-level voting for SEU correction.

B. The following steps describe the improved frame level redundancy internal scrubbing algorithm:
   1. Repeat items 1 to 3 of methodology A.
   2. Compute CRC code to be stored in the configuration memory frame header.
   3. Repeat step 4 of methodology A.
   4. Re-compute cyclic redundancy for error detection.
   5. On detection of SEU, voting is performed on the configuration frame.

C. Comparison of the results obtained with that of the existing using error correction time and energy consumption as metrics for the purpose of validation.

To the best knowledge of the researchers, there is no known single command capable of generating a random binary array of numbers MATLAB. A command (*rand*) capable of generating a random distribution of numbers with a mean of zero and deviation of 1 do exist. However, since the FPGA configuration memory consists of binary numbers (0 and 1) and based on the numbers of SRAM cells in the FPGA module, this random command was used with a limiting factor to formulate an equation capable of generating the configuration memory containing only random binary logic. Therefore, the expression used to generate the configuration memory is written as:

$$M_n = \varphi(\tau) > \eta \quad (7)$$

where,

M_n is the modules array (n= 1, 2 and 3), ϕ is a random number generator (rand) in the range of 0 and 1, ϕ was implemented as *rand* in the MATLAB script. τ is the FPGA configuration memory module dimension which is an *N* by *D* binary matrix.

η is a limiting factor whose values ranges from. A higher value of η will lead to a logic matrix with more 1 and a smaller value of η will lead to a logic matrix with more 0.

3.1 Fault Injection

With the information about the organization of the configuration memory of the FPGA and the specific commands sequence to read and write frames, any bit(s) of the configuration memory can be flip thus emulating the effect of SEU when the FPGA is exposed to radiation environment such as space. For the purpose of this research, the fault injected module is made user dependent.

The existing FLR continuously attempt to perform bit level voting to scrub against SEU on the configuration memory frame irrespective whether there is an upset or not in the configuration memory thereby causing the SRAM cells to be frequently accessed which increases the dynamic power of the device. However, the improved FLR employs CRC as an error detection scheme against SEU, meaning bit level voting is only performed when an upset is detected.

How fast SEUs are mitigated in applications such as space technology is critical because a single bit flip in the sensitive bits in the FPGA configuration memory can cause malfunction of the device or cause the device to produce faulty outputs thereby jeopardizing the satellite mission. Thus, CRC is a fast and effective error detection scheme which detects SEU before they are corrected. The time it takes to correct (scrub) these errors affects the energy consumed in the correction process since energy is a function of time.

4 RESULTS AND DISCUSSIONS

The developed algorithms were implemented in MATLAB and simulation results obtained are presented in this section. Figure 4, shows the error detection time against the number of frames.

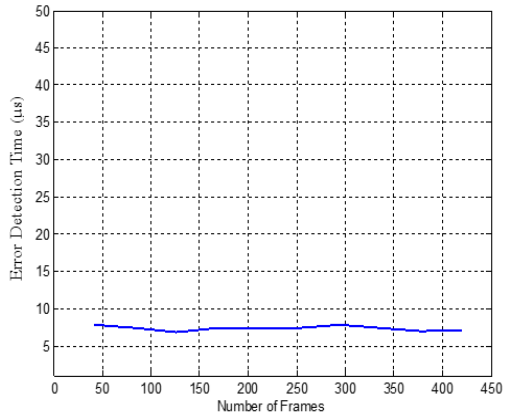

Figure 4: Error Detection Time versus Number of Frames for One Module Fault Injection.

Figure 5 shows the graph of error detection time against a number of the frame in a module, the graph was plotted using MATLAB simulation environment from the simulation parameters and results.

From Figure 4, it was observed that the time to detect an error (SEU) in the improved FLR was within the range of 7.114 and 7.333 microseconds irrespective of the number of frames in a module. As the number of frames increases from 13, 26, 39, 52, and 65 continuously, so also the number of cells in a frame increases from 42, 84, 126, 168, 210 continuously, and the time to detect error is within the range 7.114 and 7.333 microseconds for a module. This is because CRC is executed concurrently on the module frames and a 16-bit CRC executed can sufficiently detect an error in a frame of Virtex-5 FPGA. Therefore, irrespective of the number of frames the detection time is approximately constant.

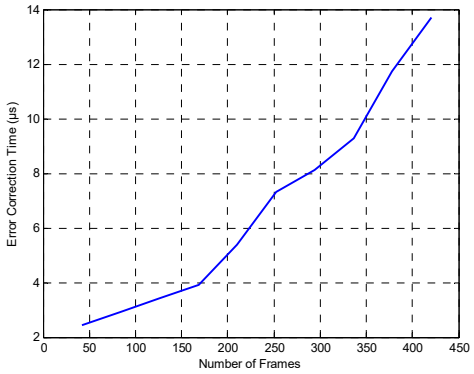

Figure 5: Error Correction time versus Number of Frames for One Module Fault Injection.

From Figure 5, it can be seen that when the number of cells in a frame and number of frames in a module increases, the time is taken to correct error also increases. This indicates that the time it will take to scrub a module is dependent on the number of injected fault and the size location of the fault injection. Because as the location of the fault injected increases, the time to scrub that area size also increases. As it was observed, for a fault injection matrix size of 12,6 24,12 36,18 48,24 60,30 continuously, the error correction time from simulated result was 2.444, 2.933, 3.422, 3.911, 5.377 microseconds respectively.

Figure 6, shows the bar chart for the energy consumed versus a number of frames in a module which was generated using MATLAB simulation environment.

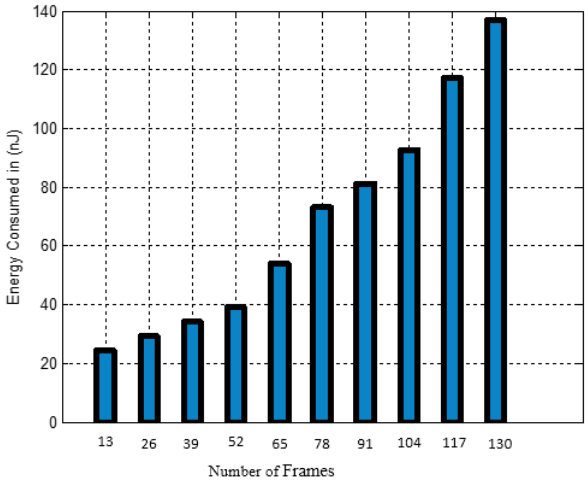

Figure 6: Energy Consumption versus a number of frames for One Module Fault Injection.

Figure 6 is the plot of energy consumed against a number of frames in one module with fault injection. It can be seen from Figure 6 that as the time to scrub increases with increase in the number of frames as well as an increase in the

number of error in one module. Likewise, the energy required to scrub the modules increases as the number of frames and injected fault also increases. This indicates that energy is a function of time at constant power. Therefore, the values obtained for energy to scrub frames is the product of the error correction time with operating power of the FPGA, where the power is the product of the FPGA operating voltage and current. It was observed from the plot that as the number of frames in a module increase from 13, 26, 39, the number of faults injected also increases as the energy required to scrub also increases from 24.44, 29.33, 34.22 Nano joules respectively.

However, in order to depict real-life scenario, SEU can also occur in any two modules (module 1 and 2, module 1 and 3, module 2 and 3) or in all the three modules (module 1, 2 and 3). Therefore, in this work the FPGA keeps in memory the original configuration of the test module for scenario where the assumption for a good bit-level voting those not hold (that is error will not occur in two or all the modules in the same frame address and at exactly the same cell position in the same scrub cycle), although this scenario is very unlikely to occur considering the enormous configuration logic bit in the FPGA and the stochastic nature of SEU.

Generally, it can be concluded that when the fault was injected in only two modules and in three modules for error detection time, error correction time and energy consumption against varying number of frames in a module, the same trend was observed as when a fault is injected in only one module. However, the magnitude of error correction time and energy consumption increases as the number of the module with fault increases from two to three. This is because the total number of injected fault in the FPGA configuration memory increases.

Figure 7 shows the comparison of Error Correction time versus Number of frames in a Module for One Module Fault Injection between Improved FLR and FLR.

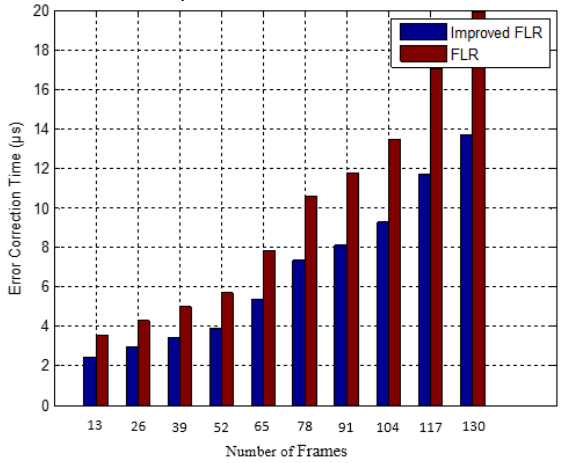

Figure 7: Comparison of Error Correction time versus Number of frames

In Figure 7, the bar chart shows the error correction time for varying module size with 13, 26, 39, 52, 65, 78, 91, 104, 117 and 130 number of frames. The bars show the comparison between the Improved FLR and the existing FLR. It was observed that when the fault was injected on a module with varying number of frames there was a reduction in the time for the Improved FLR scrubbing algorithm to correct the errors as compared to the FLR scrubbing algorithm. For the same number of injected fault, the Improved FLR scrubbing algorithm took 3.422 microseconds to correct the error for a module size with 39 frames (which is the module configuration used by the author in [2]) while the FLR scrubbing algorithm took 5 microseconds. The percentage improvement between Improved FLR and FLR is calculated to be 31.6% using the equation below.

$$\%\,improvement = \frac{FLR - improved\ FLR}{FLR} \times 100$$

The energy consumption comparison between FLR and Improved FLR is shown in Figure 8

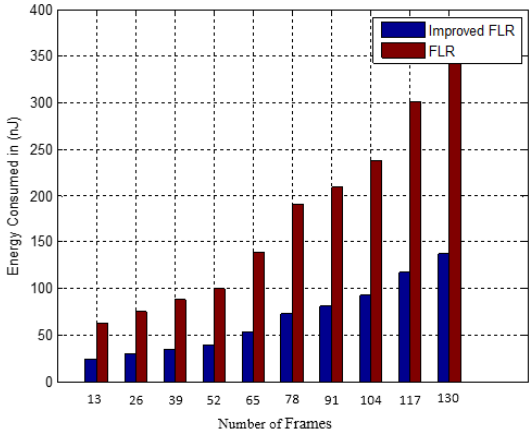

Figure 8: Comparison of Energy Consumption versus Number of frames in a Module for One Module Fault Injection between Improved FLR and FLR.

Figure 8 shows the result of the comparison in terms of energy consumption between Improved FLR and FLR scrubbing algorithm. It is observed that for a module size of 39 frames there was a reduction in the energy consumed to scrub the module when Improved FLR scrubbing algorithm was used as compared to the FLR. The percentage improvement between Improved FLR and FLR is calculated to be 61.1% using equation (8). Significant improvement was also achieved when another module size was examined as it can be clearly seen in Figure 8.

CONCLUSIONS

Frame Level Redundancy is an algorithm developed to scrub the configuration memory of SRAM-based FPGA against SEU when they are deployed in radiation environment with high energy particles such as neutron with energy in the range of Giga electron volts whereby radiation strike flips the logic state in the configuration memory causing a malfunction of the device. How fast this problem is resolved is critical as the process also impacts on the energy consumed. In other to mitigate the challenge of SEU, an improved FLR scrubbing algorithm has been developed using Cyclic Redundancy Check as an error detection technique. This was developed on a MATLAB simulation environment. The result obtained shows that when a fault is injected in one configuration memory module with thirty-nine frames, the improved FLR performed better than the FLR in terms of error correction time against SEU and the energy

consumption by 31.6% and 61.1% respectively.

ACKNOWLEDGMENTS

The authors acknowledge the Nigerian Liquefied Natural Gas (NLNG) for providing us access to the mechatronics axis of the NLNG multi-user lab where we carried out our simulation.

[1.] Jorge, T., Kastensmidt, F., & Ricardo, R. (2015). Analyzing the Effectiveness of a Frame-Level Redundancy Scrubbing Technique for SRAM-based FPGAs. *IEEE Transactions on Nuclear Science, 62*(6), 3080-3087.

[2.] Tonfat, J., Fenanda, L. K., Paolo, R., & Ricardo, R. (2015). Energy efficient frame-level redundancy scrubbing technique for SRAM-based FPGAs. IEEE Transactions on Nuclear Science, 62(6), 3080-3087.

[3.] Wirthlin, M. (2015). High-Reliability FPGA-Based Systems: Space, High-Energy Physics, and Beyond. Proceedings of the IEEE, 103(3), 379-389.

[4.] Berg, M., Poivey, C., Petrick, D., Espinosa, D., Lesea, A., LaBel, K., . . . Phan, A. (2008). The effectiveness of internal vs. external SEU scrubbing mitigation strategies in a Xilinx FPGA: Design, test, and analysis. 1-8.

[5.] Reorda, M. S., Sterpone, L., & Violante, M. (2005). Efficient estimation of SEU effects in SRAM-based FPGAs. Paper presented at the 11th IEEE International On-Line Testing Symposium, IOLTS 2005 54-59.

[6.] Jing, N., Zhou, J., Jiang, J., Chen, X., He, W., & Mao, Z. (2015). *Redundancy based Interconnect Duplication to Mitigate Soft Errors in SRAM-based FPGAs.* Paper presented at the Proceedings of the IEEE/ACM International Conference on Computer-Aided Design, 764-769.

[7.] Tambara, L. A., Tarrillo, J., Kastensmidt, F. L., & Sterpone, L. (2016). Fault-Tolerant Manager Core for Dynamic Partial Reconfiguration in FPGAs *FPGAs and Parallel Architectures for Aerospace Applications* (pp. 121-133): Springer.

[8.] Rao, P., Ebrahimi, M., Seyyedi, R., & Tahoori, M. B. (2014). *Protecting SRAM-*
Using erasure codes. Paper presented at the IEEE Design Automation Conference (DAC), 2014 51st ACM/EDAC, 1-6.

[9.] Graham, P. S., Rollins, N., Wirthlin, M. J., & Caffrey, M. P. (2003). Evaluating TMR Techniques in the Presence of Single Event Upsets. 1-7.

[10.] Eftaxiopoulos, N., Axelos, N., & Pekmestzi, K. (2016). Low latency radiation tolerant self-repair reconfigurable SRAM architecture. *Microelectronics Reliability, 56*, 202-211.

[11.] Wirthlin, M. J., Keller, A. M., McCloskey, C., Ridd, P., Lee, D., & Draper, J. (2016). *SEU Mitigation and Validation of the LEON3 Soft Processor Using Triple Modular Redundancy for Space Processing.* Paper presented at the Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, 205-214.

[12.] Sanchez, C. A., Entrena, L., & Garcia-Valderas, M. (2015). *Partial TMR in FPGAs Using Approximate Logic Circuits.* Paper presented at the 2015 15th European Conference on Radiation and Its Effects on Components and Systems (RADECS), Spain, 1-4.

[13.] Jonathan, J., Howes, W., Wirthlin, M., McMurtrey, D. L., Caffrey, M., Graham, P., & Morgan, K. (2008). *Using duplication with compare for on-line error detection in FPGA-based designs.* Paper presented at the Aerospace Conference, 2008 IEEE 1-11.

[14.] Lanuzza, M., Zicari, P., Frustaci, F., Perri, S., & Corsonello, P. (2010). Exploiting self-reconfiguration capability to improve SRAM-based FPGA robustness in space and avionics applications. *ACM Transactions on Reconfigurable Technology and Systems (TRETS), 4*(1), 1-22.

[15.] Nazar, G. L., Santos, L. P., & Carro, L. (2013). *Accelerated FPGA repair through shifted scrubbing.* Paper presented at the 2013 23rd International Conference on Field Programmable Logic and Applications (FPL), 1-6.

[16.] Wang, P., Jiang, C., Li, Z., Xue, Q., & Tian, Y. (2014). SEU Mitigation for SRAM Based on Dual Redundancy Check Method. *International Journal of Hybrid Information Technology, 7*(5), 191-200.

[17.] Wirthlin, M., & Harding, A. (2016). Hybrid Configuration Scrubbing for Xilinx 7-Series FPGAs. *FPGAs and Parallel Architectures for Aerospace Applications* (pp. 91-101): Springer.

[18.] Jacobs, A., Cieslewski, G., George, A. D., Gordon-Ross, A., & Lam, H. (2012). Reconfigurable fault tolerance: A comprehensive framework for reliable and adaptive FPGA-based space computing. *ACM Transactions on Reconfigurable Technology and Systems (TRETS), 5*(4), 1-30.

[19.] Harward, N. A., Gardiner, M. R., Hsiao, L. W., & Wirthlin, M. J. (2016). A fault injection system for measuring soft processor design sensitivity on Virtex-5 FPGAs *FPGAs and Parallel Architectures for Aerospace Applications* (pp. 61-74): Springer.

[20.] Akagic, A., & Amano, H. (2012). *A study of adaptable co-processors for cyclic redundancy check on an FPGA.* Paper presented at the 2012 International Conference on Field-Programmable Technology (FPT), 119-124.

[21.] Battezzati, N., Sterpone, L., & Violante, M. (2011). *Reconfigurable Field Programmable Gate Arrays for Mission-Critical Applications*. New York: Springer.

[22.] Akagić, A., & Amano, H. (2011). High speed CRC with 64-bit generator polynomial on an FPGA. *ACM SIGARCH Computer Architecture News, 39*(4), 72-77.

[23.] Tarrillo, J., & Kastensmidt, F. L. (2016). Power Analysis in nMR Systems in SRAM-Based FPGAs. *FPGAs and Parallel Architectures for Aerospace Applications*, 103-119.