An Improved Frame Level Redundancy Scrubbing Algorithm for SRAM based FPGA

O. E. Haruna
Centre for Satellite Technology Development, National Space Research and Development Agency Abuja, Nigeria.

K. A. Abubilal
Electrical and Computer Engineering Department, Ahmadu Bello University Zaria, Nigeria.

A. T. Salawudeen
Electrical and Computer Engineering Department, Ahmadu Bello University Zaria, Nigeria.

ABSTRACT
The use of Static Random Access Memory (SRAM) based Field Programmable Gate Array (FPGA) in critical applications has been considered a solution in space and avionics domain due to its flexibility in achieving multiple requirements such as re-programmability and good performance. However, SRAM-based FPGAs are susceptible to radiation induced Single Event Upset (SEU) that affects the functionality of the implemented design. Therefore, an improved Frame Level Redundancy (FLR) algorithm that uses Cyclic Redundancy Check (CRC) as an error detection technique for configuration memory scrubbing, is developed as a solution to mitigate SEU through upset detection and correction. Fault injection was performed on FPGA configuration memory frames on 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.

Keywords
FPGA, SRAM, Scrubbing, FLR, SEU, configuration memory, logic bit(s).

1. INTRODUCTION
SRAM FPGAs are complementary metal oxide semiconductor (CMOS) devices with special characteristic of re-configurability making them desirable for use in systems with evolving technology [1]. The use of FPGAs have 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]. Configurable FPGAs are better alternative for application specific processing in space based applications because of their flexibility and in-system re-programmability, also FPGAs are versatile devices that allow a function to be implemented by mapping it into the FPGA's pre-existing logic resources. The mapping is referred to as its configuration [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 system gates, with hundreds of millions of configuration 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 configurable logic blocks (CLBs), routing resources, input-output blocks (IOBs), and other 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. Several techniques have been proposed to make designs reliable in the presence of event upsets. Triple modular redundancy (TMR) is a technique used to provide design hardening [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]. A commonly used method to remove configuration errors is by periodic refresh of the configuration data. This is known as configuration scrubbing.

New technologies provide increasing support for configuration scrubbing [12]. Soft error 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 gets deposited inducing voltage transients to the interconnected nodes. [10]. Fault masking techniques such as Triple Modular Redundancy (TMR) are used to improve the radiation tolerance of circuits implemented in SRAM-based FPGAs. Still, it is necessary to avoid bit upset accumulation in the configuration memory with a correction mechanism to increase the reliability of the circuit [2].

2. LITERATURE REVIEW

In [2] the authors proposed an FLR scrubbing technique to mitigate SEU, the technique is based on the principle of TMR, where the scrubber executes a bit level voting without a separate scheme to detect SEUs. Authors in [6] proposed a Duplication With Recovery (DWR) technique to correct soft error in an FPGA configuration memory for bits meant for routing resources, which contribute to the majority of soft errors in FPGAs and are most vulnerable to single event upset because they consume most area and seventy to ninety percent of the configuration bits are attributed to the routing resources. In [8] the authors proposed a scrubbing scheme which reconstructs erroneous configuration memory frame based on the concept of erasure codes. The erasure codes recovers the original frame when some of the bits are flipped. The work employs a low-cost interleaved two-dimensional (I2D) parity technique to detect MBUs in the configuration memory frames of the FPGA, once an error is detected by assuming that the erroneous frame is erased, its contents are reconstructed using an erasure code by computing an exclusive-OR operation of all bits in the temporary block that was initialize with zero bits by the recovery unit, thus, the temporary block is written into the erroneous frame. Authors in [12] presented an approach to build Partial TMR circuits for FPGAs using approximate logic circuits. The work presented a TMR circuit where the two redundant copies are built as an over approximated and under approximated copy of the original copy, whereby the approximate logic circuit performs a possibly different but closely related logic function, so that it can be used as a fault tolerant technique to mask error where it overlaps with the original circuit. Then, Partial TMR can be implemented by voting among approximate logic circuits instead of exact copies of the original circuit. [13] presented an error detection technique called Duplication with Compare (DWC) where the FPGA bitstream was duplicated together with the signal nets for a full duplication operation to provide the greatest coverage for error detection and the redundant copy is stored in an external radiation memory, and comparator insertion for external system was deployed to compare with the golden copy and if any discrepancy is detected the comparator signals an error flag. Authors in [14] proposed an approach that makes FPGA devices able to self-repair SEU, whereby detection and correction are performed inside the FPGA chip by exploiting the internal readback port for an integrity check of the configuration memory using an error detection and correction (EDAC) Circuit. The EDAC codes are internally pre-computed and stored in each sensitive frame and compared to the corresponding golden reference stored in the EDAC code memory. In [15] the authors proposed a shifted scrubbing technique whenever an error occurs the scrubber starts scrubbing the associated partition where the critical bits in the configuration memory are flipped after they are been detected. The concept explored in this work was that scrubbing does not need to start at the first configuration memory frame since different regions have different concentrations of critical bits, one can find the optimum starting frame that has more upset in the region with high number of critical bits that also minimizes the time to repair, since the work was partitioned based on the sensitivity of the bits representing the mapped design. In [16] presented a technique where a dual redundancy alongside with CRC was used to mitigate SEU. A reference FPGA was used for comparison with the test FPGA and the CRC module encodes and decodes data for error detection purposes.

Authors in [17] proposed a novel hybrid configuration scrubbing for the Xilinx 7-Series FPGAs by exploiting the on-chip frame Error Correction Code (ECC). A dedicated non-configurable logic is built into the FPGA to compute a check word for each frame during configuration readback process for single bit error detection where the internal ECC circuitry computes the location of the error and corrects the upset. To detect error multi-bit error, a global CRC is provided for the entire set of frames, and a CRC is recomputed during each full scan of the configuration memory and compared against an internal global CRC. If a multi-bit error occurs that is not detected by the individual frame ECC, the recomputed CRC will differ from the global CRC signifying that an undetected error exists somewhere in the configuration memory.

From above it is evident that SEUs in the configuration memory of SRAM-based FPGA operating in radiation environment has been a challenge, and meaningful research attention has been given to mitigate this effect through various techniques to ensure data integrity. Therefore an improved FLR scrubbing algorithm for SRAM based FPGA has been developed. The algorithm reduced the time required to mitigate SEUs which will lead to quick recovery and reduced energy consumption. This implies that the algorithm ensures and guarantee the reliability of the entire system, especially in critical application such as space technology where reliability is paramount.

2.1 Single Event Upset

SEU is a form of Single Event Effects (SEE) which are 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 particle 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 Logic Block (CLB) or Look up 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 of 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.

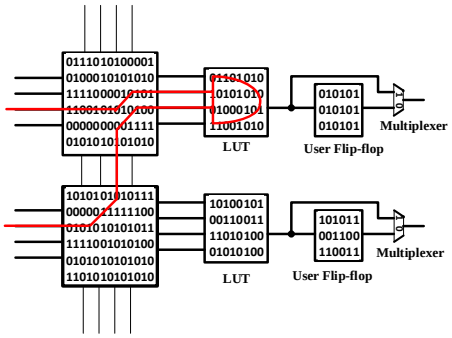

Fig. 1 Configuration Memory Used to Specify Logic and Routing [19]

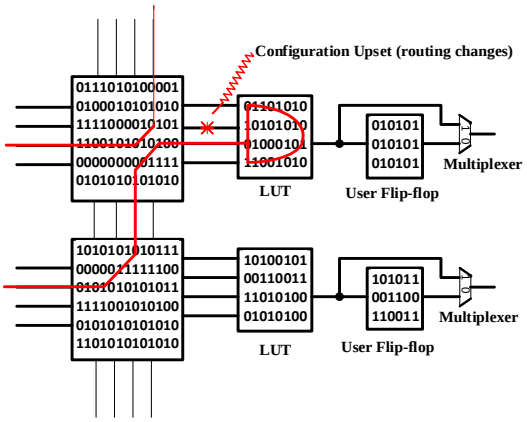

Fig. 2 Upset in Routing [19]

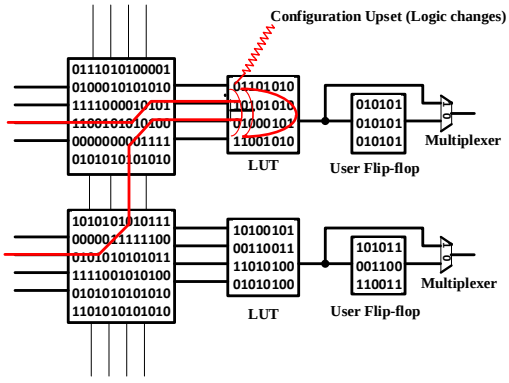

Fig. 3 Upset in Logic [19]

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 for 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 configuration memory data word with an appended r-bits redundancies. The following steps are executed (Zhang & Ding, 2011):

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

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

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:

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

where:

$x^{n-k}M(x)$ is the encoded configuration memory data

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

$Q(x)$ is the quotient

Therefore, the encoded data can be expressed as:

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

The data in the storage device is the dividend and the remainder becomes the result. Every bit in the storage device requires one exclusive-OR (XOR) and one shift operation to the left by 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 by 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 for 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]:

$$P_T = P_D + P_S \tag{4}$$

$$P_S = V_{CC} + I_{CC} \tag{5}$$

where:

V_{CC} is the voltage level

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

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 assume that the use of three modules will mainly impact the dynamic power component [23].

3. DEVELOPED SCHEME

To the best knowledge of the researchers there is no known single command capable of generating random binary array of numbers MATLAB. A command (*rand*) capable of generating random distribution of numbers with a mean of zero and deviation of 1 do exist. However, since the FPGA configuration memory consist 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 logics.

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 as $0 < \eta < 1$

Higher value of η will lead to a logic matrix with more 1 and 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.

3.2 Majority Voting Implementation

This technique relies on the generation of a Triple Modular Redundancy (TMR) design to enable voting possible and eliminating a scenario resulting to a split vote where the voter will be unable to identify the correct bit(s) from the set of bit(s) in the configuration memory frame. Therefore, majority is the greater part or more than half of the total bits under consideration which implies that it is a subset larger than any other subset. In this work, the voting is realized using the logical equation.

$$\mathrm{M_v} = \sum_{\mathrm{i=1}}^{\mathrm{length(m)}} \begin{Bmatrix} \mathrm{M_1(:,:),\ M_2(:,:),\ M_3(:,:)} \\ \text{if } \mathrm{M_1(:,:) = M_2(:,:)\ \&\ M_1(:,:) = M_3(:,:)} \\ \mathrm{M_1(:,:),\ M_2(:,:)} \\ \text{if } \mathrm{M_1(:,:) = M_2(:,:)\ \&\ M_3(:,:) \neq M_1(:,:)} \\ \text{then } \mathrm{M_3(:,:) \Leftrightarrow M_1(:,:)} \end{Bmatrix} \quad (8)$$

where,

M_v represent the voted module.

$\Leftrightarrow$ is used for element by element voting.

(:,:) is used for bit-level operation.

M_1, M_2 and M_3 are the module whose elements needs to be voted.

3.3 Energy Consumption

Since the energy consumed in the configuration memory frame during readback and scrubbing rate depends on time, therefore the following equation is used to compute the energy [2].

$$\mathrm{E_{Scrub\_frame}} = \mathrm{P_{Scrubbing}} \times \mathrm{t_{Scrub\ rate}} \quad (9)$$

where:

$\mathrm{E_{Scrub\_frame}}$ is the energy required to scrub the configuration memory frames.

$\mathrm{P_{Scrubbing}}$ is the power required to scrub the configuration memory frames.

$\mathrm{t_{Scrub\ rate}}$ is the time taken to scrub the configuration memory frames.

The power to scrub the configuration memory frame is given by [2]

$$\mathrm{P_{Scrubbing}} = \mathrm{V_{core}} \times \mathrm{I_{core}} \quad (10)$$

where:

$\mathrm{V_{core}}$ is the internal supply voltage.

$\mathrm{I_{core}}$ is the maximum current.

Figure 4 shows the implemented improved frame level redundancy algorithm flow chart.

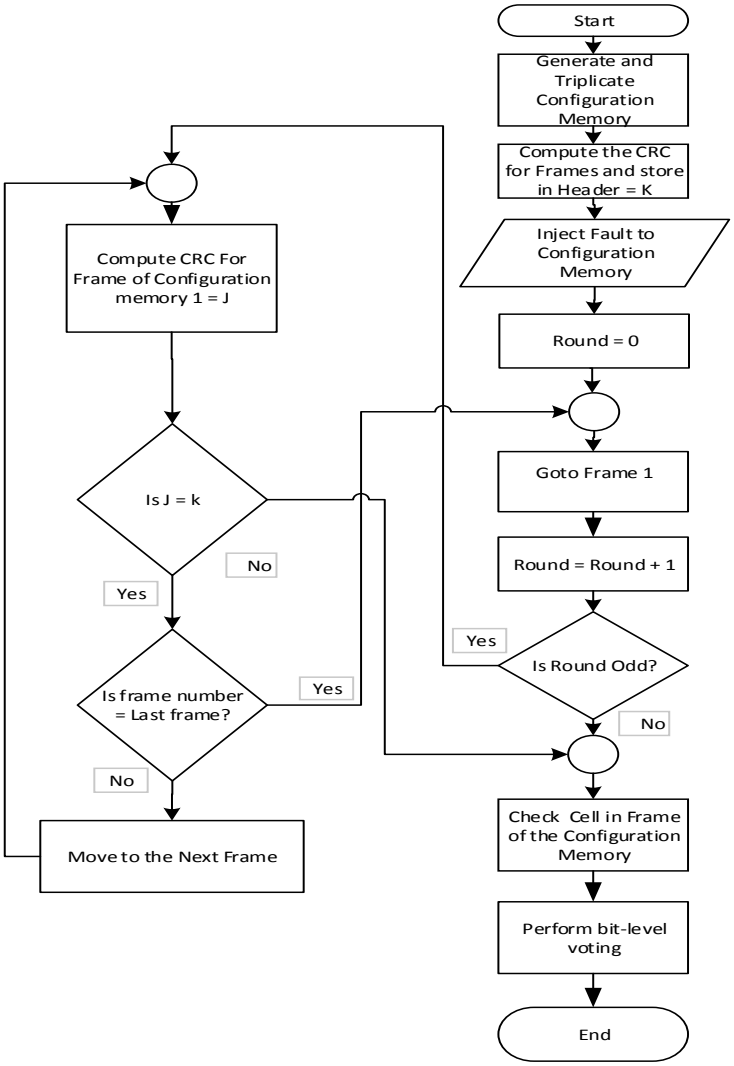

Fig. 4 Improved Frame Level Redundancy Scrubbing Algorithm.

4. RESULTS AND DISCUSSIONS

Table 1. Time and Energy Consumption for One Module Fault Injection

Virtex-5 FPGA							
Number of module with fault			1				
Specification	**Voltage (V)**		1.0				
	Current (A)		0.01				
Number of cells	**Number of frames**	**Fault Injection matrix**	**Starting and Ending cell address**	**Starting and Ending frame address**	**Error detection time (µsec.)**	**Error correction time (µsec.)**	**Energy to scrub frame(n joules)**
42	13	12,6	1,12	6,11	7.332	2.444	24.441
84	26	24,12	1,24	11,22	7.332	2.933	29.329

126	39	36,18	1,36	16,33	7.332	3.422	34.218
168	52	48,24	1,48	21,44	7.121	3.911	39.106
210	65	60,30	1,60	26,55	7.121	5.377	53.771
252	78	72,36	1,72	31,66	7.333	7.319	73.185
294	91	84,42	1,84	36,77	7.333	8.108	81.076
336	104	96,48	1,96	41,88	7.114	9.288	92.878
378	117	108,54	1,108	46,99	7.114	11.732	117.320
420	130	120,60	1,120	51,110	7.333	13.689	136.870

Table 1 shows the results of error detection time, error correction time and the energy consumed during the scrubbing process when fault was injected in only one module in the FPGA configuration memory for increase number of frame and cell size of 13 and 42 respectively as well as varying size location of injected fault. The number of configuration memory cell was continuously increased by 42, the configuration memory frame by 13 and fault injection matrix size by 12,6 for every simulation run as shown in Table 1 in other to investigate the pattern of the variation. The voltage and current specification of virtex-5 FPGA was obtained from FPGAs data sheet. The premise here is that two or three module will not have fault injected at exactly the same cell position with the same frame address within a scrub cycle. For this to effectively hold, then one module only have to be injected with fault.

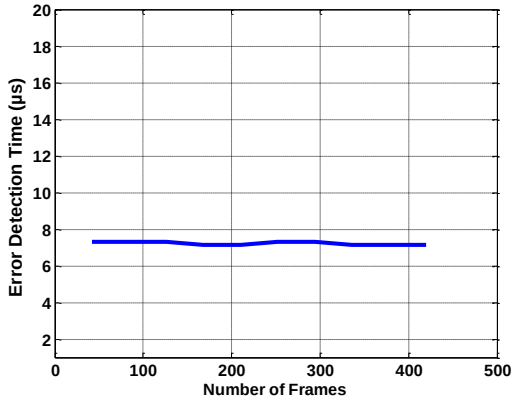

Fig. 5 Error Detection Time versus Number of Frames for One Module Fault Injection.

Figure 5 shows the graph of error detection time against number of frame in a module, the graph was plotted using MATLAB simulation environment from the simulation parameters and results obtained in Table 1.

From Figure 5, it was observed that the time to detect error (SEU) in the improved FLR was within the range of 7.114 and 7.333 micro seconds irrespective of the number of frames in a module. As the number of frames increases from 13, 26, 39, 52, and 65 continuously as shown in Table 4.4, 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 micro seconds for a module. This is because CRC is executed concurrently on the module frames and a 16-bit CRC executed can sufficiently detect error in a frame of virtex-5 FPGA. Therefore, irrespective of the number of frames the detection time is approximately constant.

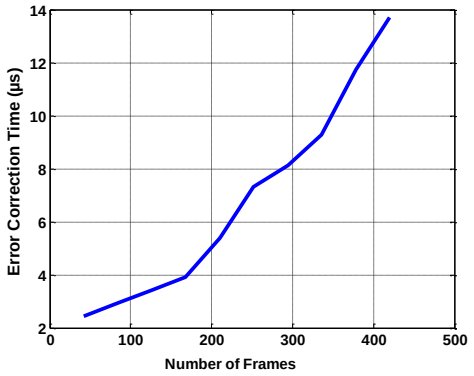

Fig. 6 Error Correction time versus Number of Frames for One Module Fault Injection.

From Figure 6, it can be seen that when the number of cells in a frame and number of frames in a module increases, the time 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 were 2.444, 2.933, 3.422, 3.911, 5.377 micro seconds respectively as shown in Table 4.4.

Figure 7, shows the bar chart for the energy consumed versus number of frames in a module. The graph was plotted using MATLAB simulation environment from the simulated parameters and results obtained in Table 1.

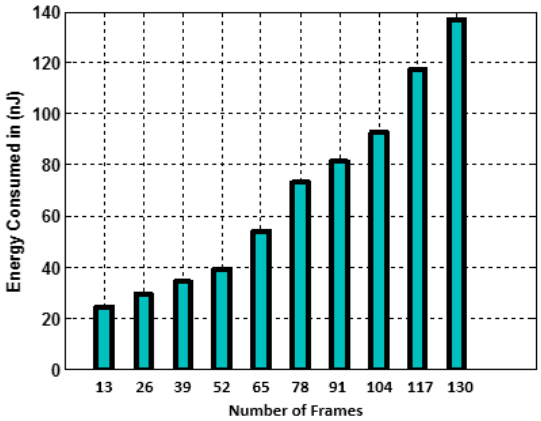

Fig. 7 Energy Consumption versus number of frames for One Module Fault Injection.

Figure 7 is the plot of energy consumed against 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 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 as shown in equation 9. Therefore, the values obtained from Table 1 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 as given in Table 1. 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 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 fault is injected in only one module. However, the magnitude of error correction time and energy consumption increases as the number of module with fault increases from two to three. This is because the total number of injected fault in the FPGA configuration memory increases.

4.1 Performance Evaluation

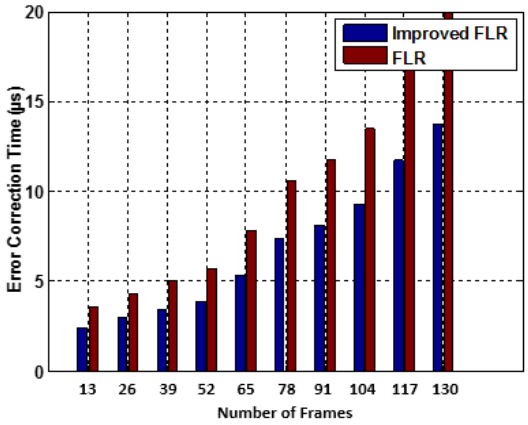

Fig. 8 Comparison of Error Correction time versus Number of frames in a Module for One Module Fault Injection between Improved FLR and FLR.

In Figure 8, 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 shows the comparison between the Improved FLR and the existing FLR. It was observed that when 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 micro seconds 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 micro seconds. The percentage improvement between Improved FLR and FLR is calculated to be 31.6% using:

$$\text{Percentage improvement} = \frac{\text{FLR - Improve FLR}}{\text{FLR}} * 100 \qquad (11)$$

This implies that the Improved FLR scrubbing outperformed the FLR scrubbing algorithm in the time taken to scrub against SEU by 31.6%. The Improved FLR was also observed to outperform the FLR for other module size.

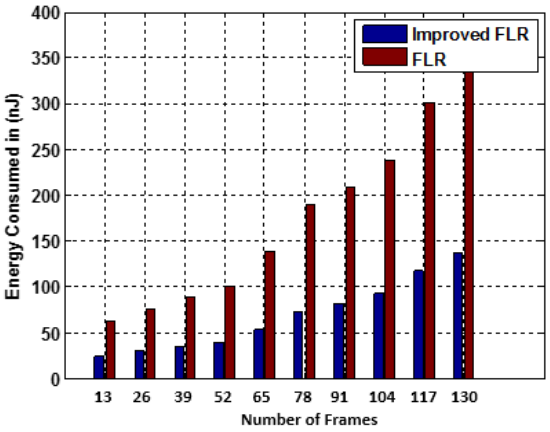

Fig. 9 Comparison of Energy Consumption versus Number of frames in a Module for One Module Fault Injection between Improved FLR and FLR.

Figure 9 shows the result for 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 (11). Significant improvement was also achieved when other module size was examine as it can be clearly seen in Figure 8.

5. CONCLUSION

SEU has become a challenge in the configuration memory of SRAM-based FPGA. How fast this problem is resolved is critical in some applications 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 (CRC) as an error detection technique. This was developed on a MATLAB simulation environment. The result obtained shows that when fault is injected in one configuration memory module, the improved FLR performed better than the FLR in terms of error correction time and energy consumption by 31.6% and 61.1% respectively. Further work can be focused on mitigating SEU in the application layer as error in some logic resources may propagate to the application layer without being detected by the configuration memory readback.

6. ACKNOWLEDGEMENTS

The authors are grateful to the Computer and Control research group of Ahmadu Bello University Zaria for their technical and professional advice in the course of this work. Also, the lead author wishes to thank his Organization National Space Research and Development Agency for the opportunity to obtain a Master's Degree.

7. REFERENCES

[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). 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-*

[9] *Using erasure codes.* Paper presented at the IEEE Design Automation Conference (DAC), 2014 51st ACM/EDAC, 1-6.

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

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

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

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

[19] 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.

[20] 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.

[21] 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.

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

[23] 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.

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