Trans. JSASS Aerospace Tech. Japan
Vol. 12, No. ists29, pp. Tf_19-Tf_25, 2014

Topics

LEO Single Event Upset Emulator for Validation of FPGA Based Avionics Systems

By Mohamed Mahmoud IBRAHIM, Kenichi ASAMI and Mengu CHO

Kyushu Institute of Technology, Kitakyushu, Japan

(Received June 24th, 2013)

This paper presents a complete design and implementation of a Single Event Upset (SEU) emulation system that can be used to inject faults Static Random Access Memory (SRAM) based Field Programmable Gate Array (FPGA). The FPGA is used to implement an avionics system for a small satellite. The fault injector emulates the expected Single Event Upset (SEU) rate as it would be in the Low Earth Orbit (LEO) of the polar orbiting satellites at inclinations close to 98° deg., and altitude of about 670 km. The emulator injects faults in the configuration bit-stream of the FPGA without stopping its operation. It makes use of the partial reconfiguration feature of today's FPGAs. This provides a facility to assess the design performance in space even if radiation testing will not be conducted before launching. Also, it simulates the expected upset rate and hence calculates the corresponding data failure rates for Triple Modular Redundancy (TMR) fault tolerant designs. The system was implemented using the Xilinx Virtex- LX50T FPGA. The FPGA suffered system failures during the fault injection test. It recovered about 50% of the failures. TMR simulation at an upset rate of 0.1 upsets (per bit per second) for a data size of 2048 bits showed that about 33% of the faults will be fully corrected.

Key Words: FPGA, SEU, Avionics Systems, TMR, Fault Tolerance

1. Introduction

Design of fault tolerant systems for space applications uses redundancy in implementation. Redundancy can be in software code, hardware units, execution times and data bits [1]. The protection techniques can be used individually or concatenated in a hybrid design. They add to the improvement of the system capability in detecting and correcting faults hence increasing its reliability, however, they also add overhead. Varieties of techniques were introduced in the literature for Hardware Fault Tolerance (HFT), Software Fault Tolerance (SFT) and Software-Implemented Hardware Fault Tolerance (SIHFT) [2-4]. These techniques include redundancy units, check pointing, recovery blocks, Error Detection And Correction (EDAC) codes, watch dog processors, control flow checking by signatures, duplicated instructions, diverse data, and others. SRAM based FPGAs make use of configuration memory scrubbing to protect their internal bitstream from bit upsets [5]. The induced bit upsets are caused by charged particles radiation such as trapped protons and electrons, Galactic Cosmic Rays (GCR) and Solar flares [6,7].

It is often required to assess the reliability of fault tolerant systems in operating conditions close to the environment where they will be used. Satellites are tested in electrical, thermal vacuum, mechanical and radiation conditions as close as possible to the target orbit. However, radiation testing at proton accelerators is often expensive, not readily available and needs complicated setups. The purpose of radiation testing is to evaluate how the design will perform in the space radiation environment. The common tests include Single Event Effects (SEE) and Total Ionization Dose (TID). The Single Event Upset (SEU) is part of the SEE where the logic values of the bits stored in the processor registers and memory cells are altered. This might lead to malfunctions and inappropriate operations. In SRAM-based FPGAs, where the design is stored in the internal SRAM after being loaded from the boot-up flash, bit alteration due to SEU can be severe. It might lead to changing the functioning logic and complete failure of the system.

In this paper we present the design of a fault injection system that can be used in emulating the bit upsets in the configuration bitstream of SRAM based FPGAs. The system emulates the SEUs as they would be found in LEO orbits at an altitude of approximately 670 km and inclination of about 98° deg. The expected upset rate is estimated using CREME96 model within the Space Environment Information System (SPENVIS) online package [8-10]. Fault lists containing bit flips at random locations of the configuration bitstream are then generated using the fault injector based on the estimated in-orbit upset rates from the CREME96 model. The purpose of this paper is to present the architecture and design of the fault injection platform that can be used in evaluating SRAM based FPGA designs. In addition to the fault injection function, the system can simulate the effects of bit upsets on the operation of Triple Modular Redundancy (TMR) protected modules. It simulates random upsets in the TMR modules and estimates the failure rate based on the simulated upsets. Thus the proposed emulator tool consists of two parts: a fault injector to inject faults in the SRAM of the FPGAs which carries the configuration bitstream and a simulator which simulates the effects of bit upsets on TMR operation.

The fault injection and simulation system was developed to target the Xilinx Virtex5 FPGA family. The fault injection function uses the SEU controller soft IP core from Xilinx [11]. Therefore the system is effective for emulating the upsets that take place in the configuration bitstream only. These upsets

target the programmable interconnects, routing information, Look Up Tables (LUT) and design logic. Faults can not be injected in the contents of the design flip flops and registers. This limitation of the fault injector is due to the fact that the used SEU controller core uses the Internal Configuration Access Port (ICAP) of the Virtex5 FPGA to access the configuration bitstream. This port can not access the contents of the flip flops and registers in the design. The flip flops and registers are usually protected using the TMR approach. The TMR simulation is thus effective in finding out the expected failure rate of the TMR protected registers and flip flops at different data sizes and upset rates. The failure in a TMR protected design takes place when the data sets from the three TMR modules are different [12,13].

There are some requirements for the proposed fault injection system to be able to operate. The fault injection is performed through an external computer that communicates with the SEU controller IP core in the Virtex5 FPGA. The generation of the fault list that would be injected takes place using MATLAB. The FPGA design must include the SEU controller IP core because this is the only way to write faults to the configuration bitstream in the FPGA SRAM. The system is designed to operate with the Virtex5 LX50 FPGA. Therefore all of the configuration frame sizes that come later in the text are based on that specific FPGA.

We hope that this work would save the proton accelerator tests and provide simple and confident test techniques to assess FPGA designs before launching into space.

In the following sections the paper introduces the SEU fault injection concept in section 2, the SEU fault injector is presented in section 3, the emulation results and discussion are presented in section 4, and the conclusion and future work are presented in section 5.

2. SEU Fault Injection Concept

Fault injection in functioning systems is a technique used to insert deliberate faults at selected and/or random units of the design to assess its sensitivities. This technique is implemented by adding additional hardware and software to the system to handle the insertion of faults, monitoring of performance and collection of results. Figure 1, shows the architecture of a fault injection system.

The design under test is interfaced to a faults insertion unit which has access to the design units where faults are to be injected. The faults vector calculation and generation unit prepares faults vectors that match the required test objectives. The fault insertion unit can be a combination of hardware and software. It handles the overriding of the normal operation into a faulty one. For example, the fault insertion unit can be a code that reads back a previously calculated value by the normal Design Under Test (DUT) code and then overwrites it with a faulty value to simulate a specific condition. The insertion can be done without stopping the main operation. In some designs it might be inevitable to interrupt the normal operation flow by suspending it and then resuming after the injection takes place. The function monitoring and control unit takes care of monitoring the operation of the DUT. It stops the DUT operation in case of noticing an emergency and provides a control path to set the DUT in specific operating modes and operation settings. The performance of the DUT is statistically analyzed to detect anomalies in normal operation as faults are injected. The feedback about how the DUT behaves while in fault injection mode is provided to the faults vector calculation and generation unit. It uses that information in generating new fault vectors. For example, the feedback statistical information might show that there is a repetitive pattern in the output when certain fault sequence is followed. The faults vector generation and calculation unit might repeat the vectors with different variations to study the statistical dependence between injected faults and output vectors. Fault monitoring and control unit also feedback the faults vector calculation and generation with information about the behavior of the DUT during the fault injection process. For example, it might be necessary to feedback the faults vector calculation and generation unit with the moments where the system completely stopped working and needed a deep reset. This information can be used in detecting the types of faults that lead to total failure.

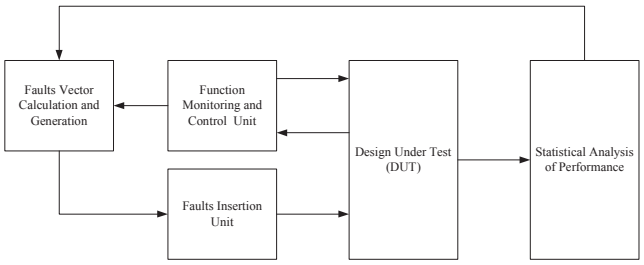

Fig. 1. Fault injection cycle. The faults are injected to the DUT and statistical results are issued as a feedback to the injecting machine for test vectors adjustment.

The SEUs which occur in space are probabilistic. Poisson distribution is used to estimate the expected number of upsets (k) which happens in the time interval (T) with an average number of upsets (μ) according to the probability density function shown in Eq. (1) [14]. The exponential distribution is used to estimate the expected time between upsets (τ) with an average number of upsets in unit time interval (λ) as shown in Eq. (2). The relationship between both distributions can be set as ($\mu = \lambda T$).

$$P(x|\mu) = (\mu^x/x!)\, e^{-\lambda} \quad (1)$$

$$P(\tau|\lambda) = \lambda e^{-\lambda\tau} \quad (2)$$

The SEU rate can be estimated using the CREME96 model [9]. The SPENVIS online tool is used in the estimations [10]. Figure 2, shows the SEU estimation for an orbit with 670 km altitude and an inclination of 98° deg. The peaks in the figure are related to upsets taking place at the South Atlantic Anomaly (SAA). The upset rate estimation is based on the values of the radiation testing of the Xilinx Virtex5 LX50 FPGA [15]. Fault Injection rate is estimated by using the per bit upset rate from the SPENVIS simulation shown in Figure 2. Faults are injected to the FPGA design using an IP core

provided by Xilinx called the SEU controller [11].

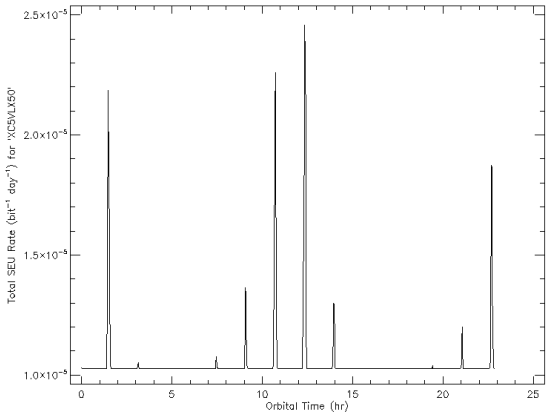

Fig. 2. SEU rate during fisrt day of flight. The peaks are orbital positions corresponding to the South Atlantic Anomaly.

3. SEU Fault Injector

The fault Injector system architecture is shown in Figure 3. The injector uses an internal hardware unit that can reconfigure the FPGA bit stream, the SEU controller. It is an IP core that is provided by Xilinx which can be controlled from outside the FPGA to produce faults in the form of bit flipping in the FPGA configuration frame. The control of the SEU controller is through serial communication over the RS232 channel to send commands to it and receive responses from it. The fault injector system contains three external computers to support its function. The fault injector computer which runs MATLAB script to generate random faults lists based on the Poisson distribution of the SEUs in the target orbit. It generates the timing at which faults will be injected which follows the exponential distribution as described earlier. Another computer is used for configuring the FPGA with the bit-stream which contains the hardware design. The design that is being used here consists of four cores of the Microblaze processor which runs together to form the avionics system of a small satellite. The cores exchange data with each other through the Fast Simplex Link (FSL) bus. This is a peer to peer direct communication between the Microblaze processors.

The function monitoring of the processors is done through sending the processors status and results of executing a simple counter program to the UART interfaces which are monitored by an external computer to collect the results and analyze them. The system runs the simulation for number of times and it generates a new fault injection vector at each time. The fault injection vector contains the bit location that will be flipped which is a random number from (0 t0 1311) and the frame number where flipping will take place which is a random number from (1 to 8662), these numbers are device specific to the Virtex5 LX50 [16]. The faults are accumulated and their effects are watched as they are injected. At the end of the injection cycle an Auto-Correction-Mode (ACM) is enabled to recover the injected faults and restore the operation of the cores. The flow chart in Figure 4, shows the test flow. The Detection-Only-Mode (DOM) is used during the accumulated fault injection. The SEU controller only monitors and reports faults when operating in the DOM.

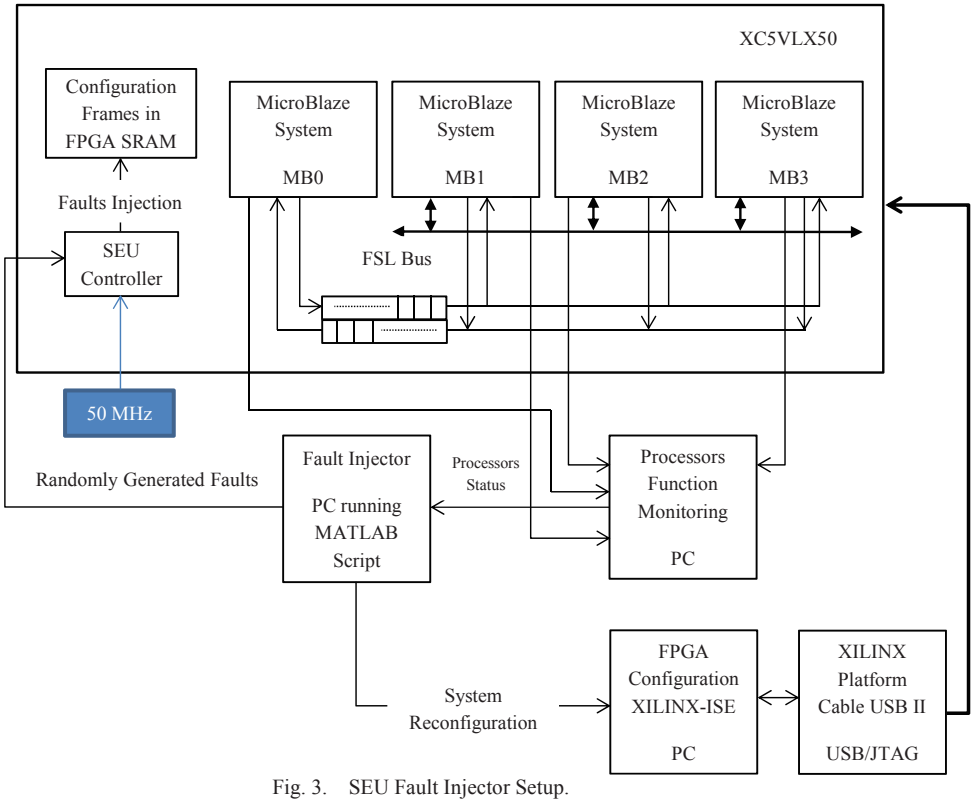

Fig. 3. SEU Fault Injector Setup.

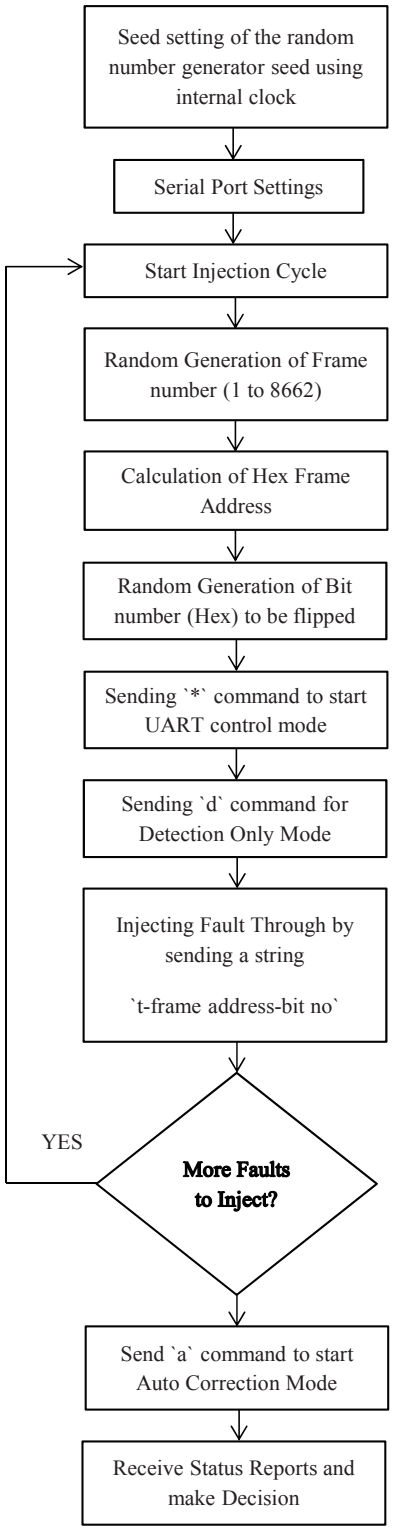

Fig. 4. Test flow Setup. The faults are injected using the commands over serial interface with the SEU controller. The results are collected over the serial interface with the processors cores. The auto-correction mode is enabled at the end of operation to recover all the injected faults.

The function of the TMR fault mitigation approach is tested through injecting faults in the data carried by the redundant modules. Each module carries the same set of data which might represent operation state code, software variables, communication message or any other form of application specific information. The TMR concept depends on voting among the data as shown in Figure 5.

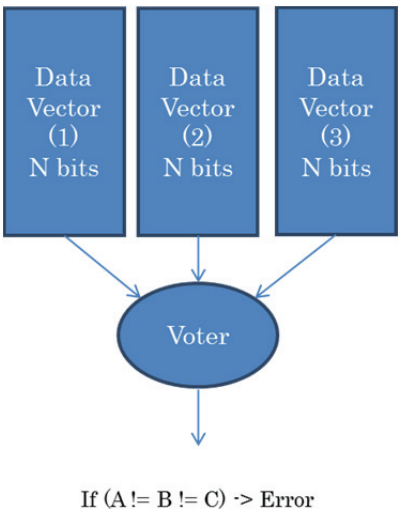

Fig. 5. TMR Concept.

The TMR approach is used in protecting the data by comparing among the data values of three functionally identical modules. In case differences among the data values exist then consensus among the values would be used to propose the most accurate value. If no consensus is found then an error is signaled. In simulating faults in the TMR data vectors of (N bits), a random fault list is generated to indicate the locations of the bits to be flipped. The data sets are then compared to each other and errors are calculated based on the comparison results as shown in the script of Figure 6.

```
% Fault Simulation
for i = 1:Total_Upsets
        Byte = ceil(random_bits(i,1)/8);
        Bit = rem(random_bits(i,1),8);
        Buffer_1(Byte) =
bitxor(Buffer_1(Byte),2^(Bit));
        Byte = ceil(random_bits(i,2)/8);
        Bit = rem(random_bits(i,2),8);
        Buffer_2(Byte) =
bitxor(Buffer_2(Byte),2^(Bit));
        Byte = ceil(random_bits(i,3)/8);
        Bit = rem(random_bits(i,3),8);
        Buffer_3(Byte) =
bitxor(Buffer_3(Byte),2^(Bit));
end
%calculation of TMR system failure rates
for i = 1 : Buffer_Size(BS_i)
if (Buffer_1(i)~= Buffer_2(i)) && (Buffer_1(i)~= Buffer_3(i)) &&
(Buffer_2(i)~= Buffer_3(i))
Unsim_Failures(j) = Unsim_Failures(j) + 1;
end
end
```

Fig. 6. TMR Simulation Script.

4. The emulation Results and Discussion

We implement the fault injection system basically to evaluate the operation sensitivity of an Multi-Processor System-on-Chip (MPSoC) avionics system currently under development by Kyushu Institute of Technology. The avionics system is designed on an SRAM based FPGA (Virtex5 LX50). The system schematic as produced by the Embedded Development Kit (EDK) package is shown in Figure 7.

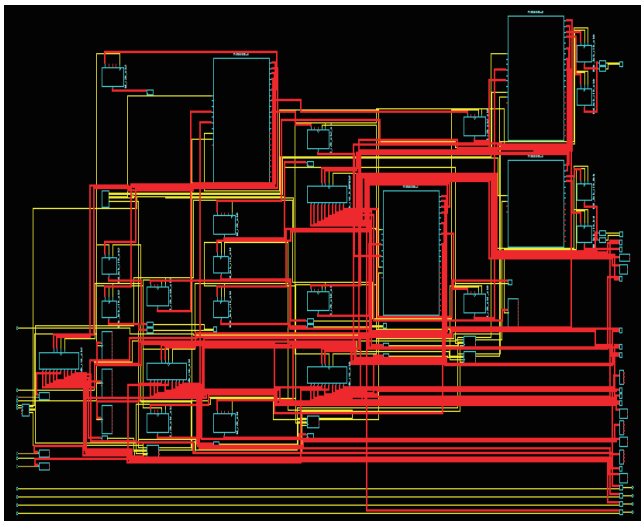

Fig. 7. EDK Schematic of MPSoC Avionics System by Kyutech.

The MPSoC contains 4 Microblaze processor systems which implement a fault tolerant architecture for a small satellite avionics platform. Assessing the performance of digital designs under radiation can be performed through circuit simulations of the critical charges and nodes, software simulations and fault injections in hardware [17-21]. We chose to implement our system using hardware fault injection. This is due to the fact that our system runs both hardware (4 Microblaze processors) and software (satellite avionics software) and we would like to assess the hardware/software integrated workability under faults. We need to see the fault injection effects on software as well as on hardware in the real working system. If we decide to simulate the hardware and software it would be a time consuming task especially if we change the design at any stage. Although it is difficult to build the system for fault injection in hardware, this difficulty is faced only once at the initial stage of building the system. Later, we can change the DUT designs without having to rebuild a new software simulator or re-simulate the change in the digital circuit design. We just change the software and hardware and download them to the FPGA and the same fault injection system would still be functioning. We think that hardware fault injection is difficult to build at the first time but it gives *flexibility* in later stages of the development cycle. Also, it enables the developer to test the hardware and software of the DUT in its integrated final form as it would operate in reality.

Figure 8, shows the results of running the fault injection campaign in the avionics system in Fig.7 with two fault lists sizes: 50 accumulated faults and 100 accumulated faults.

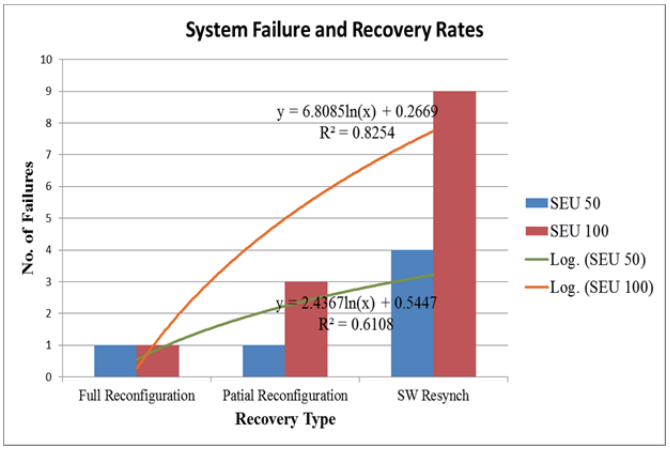

Fig. 8. Fault Injection Results.

Each fault list was generated randomly for 10 times and the results were collected for the accumulation of the faults at ach time. In the system avionics system design there are three types of correction that can take place: the full FPGA reconfiguration, the partial reconfiguration and the Software resynchronization. The full reconfiguration is the mode where the FPGA stopped working due to fault injections. The entire bit-stream of the design should be reloaded to the internal SRAM in order to restore the correct operation. The partial reconfiguration is the mode where one or more processor stopped working but not the whole system. The system can be partially reconfigured without stopping the other processors to restore the operation. The Software resynchronization is the mode where the software of the working processors need to be resynchronized to the same operation after one or more processors stopped working and then resumed again.

The results show that about 10% of the injected faults in the 50 faults batch and 10% of the 100 faults batch needed full reconfiguration. Another 10% of the faults in the 50 faults batch needed partial reconfiguration while 30% of the faults injected in the 100 faults batch needed partial reconfiguration. This means that in the 50 faults batch, only 80% of the injected faults where totally recovered through the ACM of the SEU controller without the need for partial or full reconfiguration. In the case of the 100 faults batch, 60% of the injected faults were fully recovered with no need of any reconfiguration. The software resynchronization takes place whenever a partial reconfiguration is initiated or a processor stops operation then resumes after the auto-correction mode has been enabled. The obtained results of fault injection coincides with the concepts mentioned at [22-23] that about 10% or less of the faults would lead to total system failure. This result can be viewed as a validation of the fault injector operation. Figure 9, verifies the fault injector function as it shows the plot of 50 injected faults versus the times between injections in seconds. The times between injections follow an exponential distribution. The number of faults themselves are follows Poisson distribution as stated in Eq 1, 2.

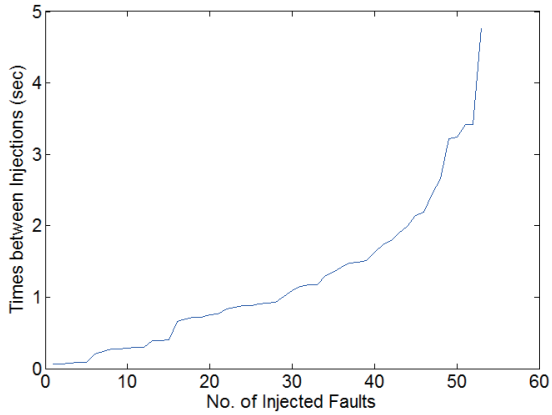

Fig. 9. Time Distribution between Injected Faults.

The TMR system was simulated at two data vector sizes, 256 bytes and 2048 bytes. We use the equation deduced in [12,13], to evaluate the TMR simulator behavior.

$$R = \frac{1}{T_c} - \frac{1}{T_c}\prod_{i=1}^{M}[3\exp(-2N_i r T_c) - 2\exp(-3N_i r T_c)] \qquad (3.)$$

Where (R) is the TMR system failure rate, (Tc) is the cycle operation time during which faults are monitored also called the scrubbing time and we chose as 1 sec, (N) is the number of bits in each TMR module and we set as 8 bits, (M) is the number of TMR groups and we set it as 256 and 2048. Figure 10, shows the results of applying the fault injection over a packet size of 2048 bytes in a TMR operation. The packet contained a random vector of data and the vector is compared between three of the operating cores after faults were injected randomly in it. The vectors are compared value by value in an TMR operation through a voter in the fourth processor. The upper curve (green) shows the failure rate as estimated by Eq. (3), while the lower curve (red) shows the simulation results, both curves saturate at the same value.

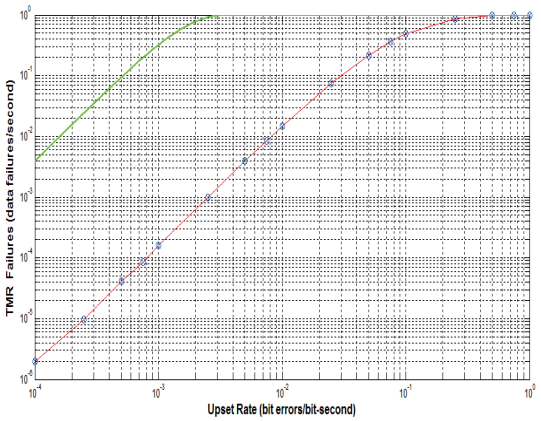

Fig. 10. TMR Failure Rate at Data vector of size 2048 bytes.

5. Conclusion and Future Work

This paper presented a fault injection emulator that can be used for injecting random faults in the FPGA bit-stream to simulate the effects of the space environment. About 10% of the injected faults in the hardware bit-stream needed full reconfiguration. In the case of data fault injection at an upset rate of 0.1 upsets per bits per second, more than 50% of the data will have residual failures. We simulated the TMR failure rates and compared it to the analytical estimation. We recommend continuing the study of the effects of faults injection on other fault tolerant designs.

References

1) Israel Koren, Mani Krishna, “Fault Tolerant Systems”, ElSEVIER 2007.
2) philip P. Shirvani, "Fault-Tolerant Computing for Radiation Environments", Ph.D thesis, Center for reliable Computing, Stanford University, June, 2001.
3) Laura L. Pullum, "Software Fault Tolerance Techniques and Implementation", Artech House, London, 2001.
4) Olga Goloubeva, “Software Implemented Hardware Fault Tolerance”, Springer 2006.
5) F.D. Lima and L. Carro, “Fault Tolerance Techniques for SRAM-based FPGAs”, Springer, 2006.
6) Daniel Hastings and Henry Garret, “Spacecraft-Environment Interactions”, Cambridge University Press (August 19, 2004).
7) Alan C. Tribble, The Space Environment: Implications for Spacecraft Design, Princeton University Press; Rev. Exp. edition (September 22, 2003).
8) Adams, J. H., Jr., Cosmic Ray Effects on MicroElectronics, Part IV, NRL Memorandum Report 5901, 1986.
9) Tylka, A.J. et al.,"CREME96: A Revision of the Cosmic Ray Effects on Micro-Electronics Code", IEEE Transactions on Nuclear Science, 44, 2150-1260 (1997).
10) SPENVIS online pacakge website: www.spenvis.oma.be
11) Ken Chapman, "New Generation Virtex-5 SEU Controller," Xilinx, Version A.2 – s4th November 2009.
12) L. Edmonds, Analysis of SEU Rates in TMR Devices, Internal Document, JPL Publication 09-6, February 2009.
13) Allen, G.; Edmonds, L.D.; Swift, G.; Carmichael, C.; Chen Wei Tseng; Heldt, K.; Anderson, S.A.; Coe, M., "Single Event Test Methodologies and System Error Rate Analysis for Triple Modular Redundant Field Programmable Gate Arrays," Nuclear Science, IEEE Transactions on, **58** no.3, pp.1040,1046, June 2011
14) Brendan Bridgford, Carl Carmichael, and Chen Wei Tseng, Single-Event Upset Mitigation Selection Guide, XAPP 987, March 18, 2008.
15) Quinn, H.; Morgan, K.; Graham, P.; Krone, J.; Caffrey, M.; , "Static Proton and Heavy Ion Testing of the Xilinx Virtex-5 Device," Radiation Effects Data Workshop, 2007 IEEE, **0** no., pp.177-184, 23-27 July 2007.
16) Xilinx, Virtex-5 FPGA Configuration User Guide, Xilinx UG191 (v3.10), 2011.
17) Pavan, P.; Tu, R.H.; Minami, E.R.; Lum, G.; Ko, P.-K.; Chenming Hu, "A complete radiation reliability software simulator," Nuclear Science, IEEE Transactions on, **41** no.6, pp.2619,2630, Dec. 1994
18) Wang Zhongming (王忠明) et al , "A software solution to estimate the SEU-induced soft error rate for systems implemented on SRAM-based FPGAs", Journal of Semiconductors, **32** no.5, 2011
19) Kafka, L.; Novak, O., "FPGA-based fault simulator," Design and Diagnostics of Electronic Circuits and systems, 2006 IEEE , pp.272,276, 18-21 April 2006
20) Bosio, A.; Di Natale, G., "LIFTING: A Flexible Open-Source Fault Simulator," Asian Test Symposium, 2008. ATS '08. 17th, pp.35,40, 24-27 Nov. 2008
21) Straka, M.; Kastil, J.; Kotasek, Z., "SEU Simulation Framework

for Xilinx FPGA: First Step towards Testing Fault Tolerant Systems," Digital System Design (DSD), 2011 14th Euromicro

22) Carl Carmichael, Michael caffrey, Anthony Salazar, “Correcting Single-Event Upsets through Virtex Partial Configuration,” XILINX, XAPP216, (v1.0), 2000.

23) SEU Strategies for Virtex-5 Devices ken chapman, XAPP864 (v2.0) April 1, 2010.