DEVELOPMENT OF AN IMPROVED FRAME LEVEL REDUNDANCY SCRUBBING ALGORITHM FOR STATIC RANDOM ACCESS MEMORY BASED FIELD PROGRAMMABLE GATE ARRAY

By

Ocholi Enoch HARUNA

Department of Electrical and Computer Engineering
Faculty of Engineering
Ahmadu Bello University Zaria, Nigeria

JULY, 2017

DEVELOPMENT OF AN IMPROVED FRAME LEVEL REDUNDANCY SCRUBBING ALGORITHM FOR STATIC RANDOM ACCESS MEMORY BASED FIELD PROGRAMMABLE GATE ARRAY

By

Ocholi Enoch HARUNA, B.Eng. (ATBU, 2008)

P14EGEE8055

A DISSERTATION SUBMITTED TO THE SCHOOL OF POSTGRADUATE STUDIES, AHMADU BELLO UNIVERSITY, ZARIA

IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE AWARD OF A MASTER OF SCIENCE (M.Sc) DEGREE IN ELECTRONICS ENGINEERING

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

FACULTY OF ENGINEERING

AHMADU BELLO UNIVERSITY, ZARIA

NIGERIA

JULY, 2017

DECLARATION

I HARUNA Ocholi Enoch, hereby declare that the work in this dissertation entitled **"**Development of an Improved Frame Level Redundancy Scrubbing Algorithm for Static Random Access Memory Based Field Programmable Gate Array" has been carried out by me in the Department of Electrical and Computer Engineering, Ahmadu Bello University, Zaria. The information derived from literature has been duly acknowledged in the text and a list of references provided. No part of this Dissertation was previously presented for another degree or diploma at this or any other institution.

Haruna Ocholi Enoch _____ _____

Signature Date

CERTIFICATION

This dissertation entitled "DEVELOPMENT OF AN IMPROVED FRAME LEVEL REDUNDANCY SCRUBBING ALGORITHM FOR STATIC RANDOM ACCESS MEMORY BASED FIELD PROGRAMMABLE GATE ARRAY" by Ocholi Enoch HARUNA meets the regulations governing the award of degree of Master of Science (MSc) in Electronics Engineering of the Ahmadu Bello University, and is approved for its contribution to knowledge and literary presentation.

Dr. K. A. Abubilal (Chairman, Supervisory Committee)	_____ Signature	_____ Date
Dr. S. Man-Yahaya (Member, Supervisory Committee)	_____ Signature	_____ Date
Dr. Y. Jibril (Chairman, Supervisory Committee)	_____ Signature	_____ Date
Prof. S. Z. Abubakar (Dean Postgraduate School)	_____ Signature	_____ Date

DEDICATION

This research work is dedicated to God Almighty, my wife, daughter and my entire family.

ACKNOWLEDGEMENT

I wish to express my profound gratitude to Almighty God, The most gracious and merciful Lord. Despite the numerous efforts put into this research work, it would not have been possible without the help of Almighty God, the fountain of all wisdom, knowledge and understanding. Lord thanks for your faithfulness.

My gratitude goes to my supervisor Dr. K.A. Abubilal, for his efforts and supervision towards the success of this work. The completion of this work would not have been possible without your participation, assistance and advice. Thank you very much sir. I am grateful to my co-supervisor Dr. S. Man-Yahaya for his efforts, and understanding that helped me in the completion of this research. Thank you very much sir. My deep appreciation goes to the Computer & Control Research Group for their keen interest, suggestions and constructive criticisms during all the stages of my presentation in the group.

I am grateful to all the lecturers of Electrical and Computer Engineering Department, Ahmadu Bello University, namely: Prof. B. G. Bajoga, Prof. B. Jimoh, Dr. A. M. S. Tekanyi for his assistance and support, Dr. E. A. Adedokun for his support and encouragement, Dr. S. M. Sani for his assistance, Dr. A. D. Usman and Dr. S. Garba, Dr. Y. Jibril, Dr. T.H Sikiru, Engr.F. Sadiku, Engr. M. J. Mu'azu, Engr. A. I. Abdullahi, Engr. E. A. Gbenga, Engr. Bashir Sadiq and Engr. Salawudeen A. Tijjani for his assistance, support and guidance, Engr. Olaniyan Abdulrahman and most especially, those whose names could not be mentioned. My appreciation also goes to Abdullahi Tukur for all the administrative support towards me and the entire students of the Department of Electrical

and Computer Engineering. I also express my profound appreciation to Prof. M. B. Mu'azu for his support, guidance, assistance and attention in the course of this research.

My gratitude goes to my special friends, Atuman G. Joel, Alhassan, and Engr. Suleiman Hussein, Ibrahim Abdulwahab, Magaji Suleiman, Wasiu Ajadi, Jerry (Daddy J), Malachi E., Peter, Oyibo Prosper, Engr. Ovie, Muktar Abubakar, Mr Braimo, Ore-ofe Ajayi, Kelechi Okogwu, Caleb Ishaya Witta, Gideon Rimamfate (5Huns), Moses Uchendu. Their continuous support and contributions toward the success of this work would never be forgotten, may God reward them and strengthen our friendship may God bless and reward you all. Amen.

Above all, I am most grateful to my wife and daughter, Mrs Jessica Haruna & Michelle Haruna, for their prayers, patience, understanding, advice, unconditional and endless love, you people are my world. I am overly indebted to my parents Mrs Salome Haruna and in loving memory of my Dad Mr ENOCH ADAMA HARUNA your endless love and support, kind and understanding personality will always be appreciated. Thank you very much. To my siblings, Mrs Aboh J. Akoh, Mrs Ojoma Maipyen, Onu Haruna, Ajifa Haruna and Joseph Idih,I owe you all a big thank you. To my cousins, Marvellous Okoh, Stephen Adejo, Okpe Okoh, in loving memory of Mrs, AJUMA ZAKKA, Mrs Ilebaye jakawa, Mrs Endurance Bature, Okpanachi Idih, Idoko Idih, Ogaji Idih, I am highly grateful for your constant support. Last but not the least, a big thank you goes to my inlaws, the Emmanuel Dabis family, Ernest Dabis family, my uncle the Emmanuel Haruna's family, my aunt the Eleojo's Okoh family, Mrs Deborah Haruna, the director of CSTD Dr. Spencer Onuh, the head of NASRDA Ground receiving Station Engr. Chichebe Akachukwu, Nani Karaye, Nanpon Zwandor, Simon Chundusu, Joshua Yakubu, Gideon

Choji, Abraham Okita, Jerry Zaky and all those that have directly and indirectly contributed to the success of this work. May God bless you all.

Ocholi Enoch HARUNA
July, 2017.

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. This research presents the development of an improved Frame Level Redundancy (FLR) Scrubbing algorithm for SRAM-based FPGA to mitigate against radiation-induced SEU. The improved FLR uses Cyclic Redundancy Check (CRC) as an error detection technique to enable configuration memory scrubbing 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 R2013a. 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.

TABLE OF CONTENTS

DECLARATION i

CERTIFICATION ii

DEDICATION iii

ACKNOWLEDGEMENT iv

ABSTRACT vii

LIST OF FIGURES xi

LIST OF TABLES xiii

LIST OF ABBREVIATIONS xiv

CHAPTER ONE: INTRODUCTION

1.1 Background of the Study 1

1.2 Motivation 4

1.3 Significance of Research 4

1.4 Statement of Research Problem 5

1.5 Aim and Objectives 6

1.6 Dissertation Organization 6

CHAPTER TWO: LITERATURE REVIEW

2.1 Introduction 7

2.2 Review of Fundamental Concepts 7

2.2.1 Field Programmable Gate Array Architecture 7

2.2.2 Single Event Upset 19

2.2.3 Total Ionization Dose 25

2.2.4 Radiation Environment 28

2.2.5 FPGA Single Event Upset Scrubbing Techniques 30

2.2.6 Triple Modular Redundancy 35

2.2.7 Frame Level Redundancy Scrubbing 37

2.2.8 Cyclic Redundancy Check 40

2.2.9 Single Event Upset Detection and Correction Time 42

2.2.10 Power Consumption 43

2.2.11 Fault Injection **45**
2.3 Review of Similar Works **47**
CHAPTER THREE: MATERIALS AND METHODS
3.1 Introduction **61**
3.2 Generation of Field Programmable Gate Array Configuration Memory **62**
3.3 Configuration Memory Triplication **63**
3.4 Methodology for a Fault Injection Campaign **64**
3.5 Majority Voting Implementation **67**
3.5.1 Frame Level Redundancy Scrubbing Process **68**
3.6 Majority Voter **71**
3.7 Cyclic Redundancy Check **73**
3.7.1 Cyclic Redundancy Check Frame Execution **73**
3.8 Improved Frame Level Redundancy Scrubbing Algorithm Implementation **74**
3.9 Energy Consumption **76**
CHAPTER FOUR: RESULT AND DISCUSSION
4.1 Introduction **77**
4.2 Fault Injection Result **77**
4.3 Results of One Module Fault Injection **82**
4.3.1 Error Detection versus Module Size **83**
4.3.2 Error Correction versus Module Size **85**
4.3.3 Energy Consumption versus Number of Frame **86**
4.4 Comparison of Improved FLR and Existing FLR Scrubbing Algorithm **96**
CHAPTER FIVE: CONCLUSION AND RECOMMENDATION
5.1 Introduction **101**
5.2 Conclusion **101**
5.3 Significant Contributions **102**
5.4 Limitation **102**
5.5 Recommendations for Further Work **102**
REFERENCES **104**
APPENDIX A
MATLAB M-FILE “CONFIGURATION MEMORY GENERATION” **112**

APPENDIX B

MATLAB M-FILE "FAULT INJECTION" 113

APPENDIX C

MATLAB M-FILE “CYCLIC REDUNDANCY CHECK” 117

APPENDIX D

MATLAB M-FILE “BIT-LEVEL VOTING” 118

LIST OF FIGURES

Figure 2.1: Conceptual Layers of a Field Programmable Gate Array 8

Figure 2.2: Generic Structure of an FPGA 9

Figure 2.3: SRAM-Controlled Programmable Switch 11

Figure 2.4: General Structure of an FPGA Bitstream 13

Figure 2.5: Configuration Memory Floor Plan of Virtex-5 FPGA 14

Figure 2.6: Schematic of a 6T SRAM Cell 18

Figure 2.7: Illustration of Configuration Memory 22

Figure 2.8: Configuration Memory used to specify Logic Routing 22

Figure 2.9: Upset in Routing 23

Figure 2.10: Upset in Logic 23

Figure 2.11: Effects of Neutrons on Semiconductor Device 27

Figure 2.12: Scrubbers Classification 32

Figure 2.13: Scrubbing Mechanism 34

Figure 2.14: Implementation on an FPGA TMR 36

Figure 2.15: Frame Level Redundancy Scrubbing Flow Chart 39

Figure 3.1: Snippet of Configuration Memory Triplication Implementation 64

Figure 3.2: FPGA Configuration Memory Fault Injection 65

Figure 3.3: Bit-level Majority Voting in an FLR Scrubbing Technique 70

Figure 3.4: Flow Chart of a Voting Process 70

Figure 3.5: Snippet of bit-level Voting 72

Figure 3.6: Snippet of Cyclic Redundancy Check Implementation 73

Figure 3.7: Improved Frame Level Redundancy Scrubbing Flow Chart 75

Figure 4.1: Single Event Upset in 1 FPGA Module 79

Figure 4.2: Single Event Upset in 2 FPGA Modules 80

Figure 4.3: Single Event Upset in 3 FPGA Modules 81

Figure 4.4: Error Detection time versus Number of Frames for 1 Module Fault Injection 84

Figure 4.5: Error Correction time versus Number of Frame for 1 Module Fault Injection 85

Figure 4.6: Energy Consumption versus number of frames for 1 Module Fault Injection 87

Figure 4.7: Error Detection Time versus Number of Frames for 2 Module Fault Injection 90

Figure 4.8: Error Correction Time versus Number of Frame for 2 Module Fault Injection 90

Figure 4.9: Energy Consumption versus Number of frames for 2 Module Fault Injection 91

Figure 4.10: Error Detection time versus Number of Frames for 3 Module Fault Injection 94

Figure 4.11: Error Correction time versus Number of Frame for 3 Module Fault Injection 94

Figure 4.12: Energy Consumption versus Number of Frame for 3 Module Fault Injection 95

Figure 4.13: Comparison of Error Correction time versus Number of in a Module for 1 Module Fault Injection between Improved FLR and FLR 98

Figure 4.14: Comparison of Energy Consumption versus Number of in a Module for 1 Module Fault Injection between Improved FLR and FLR 99

LIST OF TABLES

Table 2.1: Number of Frames per Resources 15

Table 2.2: Configuration Bitstream Structure 15

Table 3.1 Majority Voter Truth Table 71

Table 4.1 Fault Injection Parameters for 1 Module 78

Table 4.2 Fault Injection Parameters for 2 Modules 79

Table 4.3 Fault Injection Parameters for 3 Modules 80

Table 4.4: Time and Energy Consumption for 1 Module Fault Injection 83

Table 4.5: Time and Energy Consumption for 2 Module Fault Injection 89

Table 4.6: Time and Energy Consumption for 3 Module Fault Injection 93

Table 4.7: Time and Energy Consumption Comparison for 1 Module Fault Injection 97

LIST OF ABBREVIATIONS

Acronyms	Definition
ASICs	Application Specific Integrated Circuits
BCH	Bose and ray-ChaudHuri
BRAM	Block Random Access Memory
CLB	Configurable Logic Block
CMOS	Complementary Metal Oxide Semiconductor
CRC	Cyclic Redundancy Check
DMR	Dual Modular Redundancy
DSPB	Digital Signal Processing Block
DWC	Duplication with Compare
DWR	Duplication with Recovery
ECC	Error Correction Code
EDAC	Error Detection and Correction
EEPROM	Electrically Erasable Programmable Read Only Memory
FLR	Frame Level Redundancy
FPGA	Field Programmable Gate Array
FIFO	First In First Out
GP	Generator Polynomial
ICAP	Configuration Access port
ICs	Integrated Circuits

IOB	Input-Output Block
I2D	Interleaved Two Dimensional
LEO	Low Earth Orbit
LUT	Look Up Tables
MBU	Multiple Bit Upset
MOSFET	Metal Oxide Semiconductor Field Effect Transistor
nMR	n-Modular Redundancy
PIP	Programmable Interconnect Point
PR	Partial Reconfiguration
SAA	South Atlantic Anomaly
SEE	Single Event Effects
SEU	Single Event Upset
SET	Single Event Transient
SRAM	Static Random Access Memory
TID	Total Ionizing Dose
TMR	Triple Modular Redundancy
WL	Word Line

CHAPTER ONE

INTRODUCTION

1.1 Background of the Study

Static Random Access Memory (SRAM) based Field Programmable Gate Arrays (FPGAs) are Complementary Metal Oxide Semiconductor (CMOS) devices with special characteristic of re-configurability making them desirable for use in systems with evolving technology (Jorge *et al.*, 2015). 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 (Wirthlin, 2015). They are attractive to critical applications due to their high performance, power consumption, and reconfiguration capability (Tonfat *et al.,* 2015), and can be re-configured in the field, design updates can be performed while the device is still operational (Kastensmidt & Rech, 2016). 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 (Li, 2012), therefore, there is great interest in exploiting these benefits in space and other radiation environments (Wirthlin, 2015). 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 (Berg *et al.*, 2008). In SRAM based FPGAs, the mapped circuit is totally controlled by the configuration memory, which is composed of SRAM cells (Reorda *et al.*, 2005). A modern generation FPGA have tens of thousands to millions

system logic gates, with hundreds of millions of configuration bits, dominating the SRAM cells in the device (Jing *et al.*, 2015). Similar to memory devices, FPGAs have high density of transistors and interconnect wires (Stott *et al.*, 2008).

While SRAM-based FPGAs offer several advantages for critical-based operations, they are sensitive to Single Event Upsets (SEUs) (Graham *et al.*, 2003). When a fault changes the state of an SRAM cell, this is referred as SEU (Tambara *et al.*, 2016). In other words, SRAM-based FPGAs are more prone to soft errors since a radiation strike in the configuration memory has a permanent effect on the functionality of the mapped design (Rao *et al.*, 2014). 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 (Graham *et al.*, 2003).

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 (Tonfat *et al.*, 2015). Therefore, due to small device geometry, a single particle strike might affect multiple adjacent cells in a memory array resulting in a Multiple Bit Upset (MBU). In

addition, increasing transistor count per chip (Moore's law), aggressive transistor downscaling, and reduced operating voltage result in an exponential growth in soft error of digital circuits in the past years, considering the proliferation use of FPGA devices in various safety and mission-critical applications (Rao *et al.*, 2014). 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 (Eftaxiopoulos *et al.*, 2016). Soft errors in the configuration memory bits of SRAM based FPGAs have a persistent effect and they remain until the original configuration is rewritten (Jorge *et al.*, 2015).

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 (Wirthlin *et al.*, 2016). In SRAM FPGAs, SEUs can affect the configuration and provoke errors that remain until the device is reconfigured. 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 (Sanchez *et al.*, 2015). 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. (Eftaxiopoulos *et al.*, 2016). Fault masking techniques such as 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 (Tonfat *et al.*, 2015).

1.2 Motivation

The SRAM-based FPGA are becoming popular in space and avionic industry because of their large logic resources, re-programmability and broad applicability. The programmability after deployment makes it possible to introduce design update, correct design mistakes thereby extending the system lifetime. The amount of data collected by space vehicle, onboard data processing capabilities enables much data to be interpreted and compressed onboard the space system before transmitting the result to ground station(s) or other space system thus, reducing data transmission requirement. However, radiation strike from high energetic particles in space environment causes the logic state of the SRAM-based FPGA configuration memory to flip resulting to SEU which modifies the function of the FPGA intended design causing the FPGA or system to produce faulty output which can jeopardize the integrity of a system or mission. Therefore, configuration memory scrubbing and some form of reliability are required as solution to address this issue.

1.3 Significance of Research

SEU is a major issue that occurs in processing and storage devices such as FPGA when used in application critical environment like space. SEU is the flipping of logic bit of the configuration memory of FPGA from one state to the other when struck by radiation. This compromises the data integrity as an upset in the configuration memory changes the functionality of the configured circuitry or design which consequently causes malfunction

of the FPGA and could lead to permanent fault when accumulated. Configuration memory scrubbing provides a solution to this problem by mitigating against SEUs and the time to detect and correct SEU is highly critical which has a way of impacting on the energy consumption. There is need to work on a reliable technique that effectively detect and correct SEU within a short period. Therefore, this research work offers a Frame Level Redundancy (FLR) scrubbing algorithm for SRAM based FPGA that employs CRC as an error detection technique to address this issue. This is the reason why this research is significant to carry out in order to discover the improvement of the technique used.

1.4 Statement of Research Problem

A variety of well-known and proven SEU mitigation techniques has been applied to FPGA-based systems and demonstrated in radiation environments. Whenever SRAM-based FPGAs are used, upsets in the configuration memory become a major dependability threat, and must be removed as soon as possible through a process called scrubbing. However, most of the scrubbing techniques impose an overhead in terms of large area, energy consumption and a longer time to correct error. These are some of the problems that arises as a result of mitigating SEU in SRAM-based FPGAs deployed in radiation environment. Therefore, there is need to introduce a technique to explore the trade-off in terms of area, power consumption error correction time and reliability in order to avoid functional failure of the device. As a result of redundancy introduced, this research developed a modified frame level redundancy algorithm as a scrubbing technique to mitigate SEU in SRAM-based FPGAs configuration memory.

1.5 Aim and Objectives

The aim of this research work is to develop an improved FLR scrubbing algorithm for SRAM based FPGA.

The objectives of the research are as follows:

A. Replicate and implement the FLR scrubbing algorithm of Tonfat *et al.,* (2015) used to mitigate SEU in SRAM-based FPGA devices.
B. Develop an improved FLR scrubbing algorithm using CRC as a detection algorithm for SEU to mitigate its effect in FPGA devices.
C. Compare of the performance results of the improved FLR scrubbing algorithm with that of the existing FLR scrubbing algorithm used by Tonfat *et al.,* (2015) in terms of error correction time and energy consumption.

1.6 Dissertation Organization

The organization of this dissertation report is as follows: Chapter one presents the general background of the study; in Chapter two, review of fundamental concepts pertinent to the research work and detailed review of similar works are presented; in Chapter three, detailed explanation of methods and materials are presented; in Chapter four, results and discussions are presented; in Chapter five, conclusions and recommendations are presented; then finally, all the references quoted in this dissertation report and appendices are provided.

CHAPTER TWO

LITERATURE REVIEW

2.1 Introduction

This section comprises of the overview of fundamental concepts and the review of similar works which serve as a guide towards achieving the set aim and objectives.

2.2 Review of Fundamental Concepts

In this section, concepts fundamental to the research work are reviewed. These fundamental concepts regarding the research work are discussed below.

2.2.1 Field Programmable Gate Array Architecture

FPGA can be divided generally into two layers, namely: application layer and configuration layer which is also referred to as the configuration memory layer. The application layer includes the logic and memory elements managed by user's application and configuration layer includes the logic bits and memory elements that allows configuring the logic and routing resources in the application layer (Herrera-Alzu & Lopez, 2013). Figure 2.1 shows a conceptual layers of a Field Programmable Gate Array.

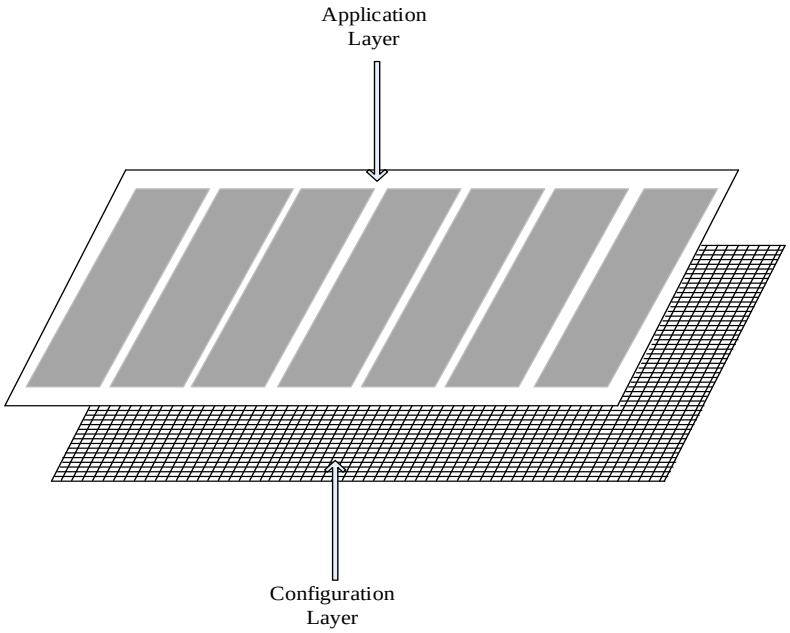


Figure 2.1: Conceptual Layers of a Field Programmable Gate Array (Siegle *et al.*, 2015).

2.2.1.1 Application Layer

The application layer comprises the logic, memory and input/output resources for user application. The layer implements Configurable Logic Blocks (CLBs), which can be configured to implement any user sequential or combinational circuit and provides the basic logic and storage functionality for a target application design. FPGAs are typically made up of highly configurable logic blocks containing Look Up Tables (LUTs) that define logic functions and registers used for sequential logic (Johnson & Wirthlin, 2009). Each CLB in turn consists of several slices, and each slice contains LUTs, registers and carry logic (Herrera-Alzu & Lopez, 2013).

2.2.1.2 Configuration Layer

The configuration layer comprises the configuration memory otherwise known as bitstream and associated access ports and control logic. The layer consist of access ports for reading and writing from/to configuration memory The configuration port is internally

connected to a built-in circuitry that decodes the configuration data providing read or write access to the configuration memory. The functionality of an FPGA is dependent on the contents of its configuration memory (Herrera-Alzu & Lopez, 2013). When considering SRAM-based FPGAs, it is possible to identify two types of memory, the one storing the configuration bitstream that defines the functionality performed by the reconfigurable cell and the one storing the application data being processed (Bolchini *et al.*, 2011).

2.2.1.3 Generic Structure of a Field Programmable Gate Array

Figure 2.2 shows a Virtex FPGA that features a regular architecture that comprises an array of CLBs surrounded by programmable Input/Output Blocks (IOBs), all interconnected by a routing resources. The routing resources permits the Virtex family to accommodate complex designs. Virtex FPGAs are SRAM based and are customized by loading configuration data into internal memory cells (Habinc, 2002).

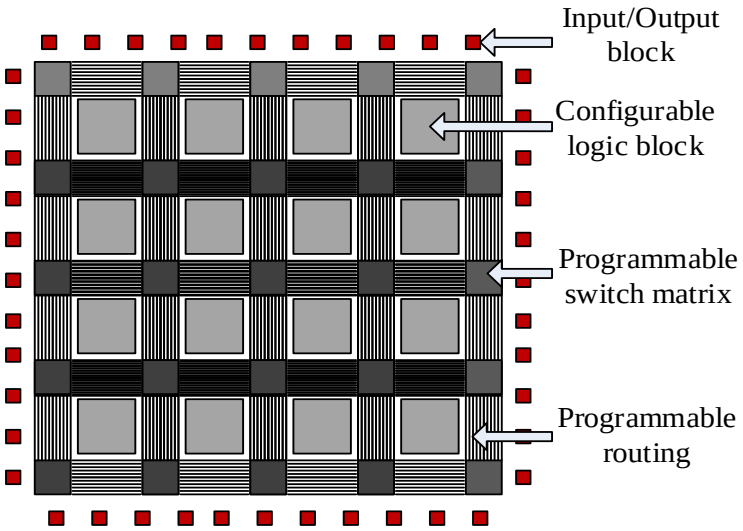


Figure 2.2: Generic Structure of an FPGA (Ottavi *et al.*, 2008).

A. Input/Output Block: The input/output blocks implement the interface between the pins of the device, the interconnections and logic blocks structure of the FPGA

architecture matrix. This interface provides the other routing resources used on the frontiers of the device with the external world and allows the connectivity of the FPGA architecture with the external signals from other devices. In addition, within the input/output blocks there are several flip-flops (Battezzati *et al.*, 2011).

B. Configurable Logic Block: The granularity is defined as the number of Boolean functions implemented by the logic or with the total number of transistor. In several FPGA architectures it is difficult to distinguish between the interconnections and the logic blocks given their connectivity. Therefore, FPGA can be classified into two possible categories namely (Battezzati *et al.*, 2011):

1. **Fine granularity:** This is also called fine-grain logic blocks which consist of interconnected elements and is formed connecting logic gate (especially NAND gate since NAND gates can be used to implement all other logic gates) to the multiplexer (as one of the elements used as programmable switch) in order to create the desired logic function either combinational or sequential.

2. **Coarse granularity:** The coarse-grain logic blocks consist of a higher number of components that is logic resources per block resources with respect to the fine-grain logic blocks, principally including multiplexers, NAND gates, and LUTs for implementing any desired Boolean functions.

C. Programmable Switches: The switch blocks are switch matrices where several Programmable Interconnect Points (PIPs) (i.e. pass transistor), called routing segments controlled by the configuration memory are available (Sterpone *et al.*, 2005). Each switching element of an FPGA architecture consists of a variable number of PIPs. The

switch blocks consist of various kinds of PIPs serving as switches to connect the horizontal and vertical wires (Jing *et al.*, 2015). Considering a switch block, the set of PIPs defines the connection of each wiring segment. In the case of SRAM technology it has five transistors that form the SRAM cell and a pass transistor creating the contact, the wiring segments controlled by the pass transistor are connected or not, depending on the logic value contained into the configuration memory cell (Battezzati *et al.*, 2011).

Figure 2.3 shows an SRAM-controlled programmable switch, showing two applications of SRAM cells for controlling the gate nodes of pass-transistor switches and to control the select lines of multiplexers that drive logic block inputs. The figure gives an example of the connection of one logic block (represented by the AND-gate in the upper left corner) to another through two pass-transistor switches, and then a multiplexer, all controlled by SRAM cells. FPGA uses pass-transistors or multiplexers or both depends on the particular product generation (Brown & Rose, 1996).

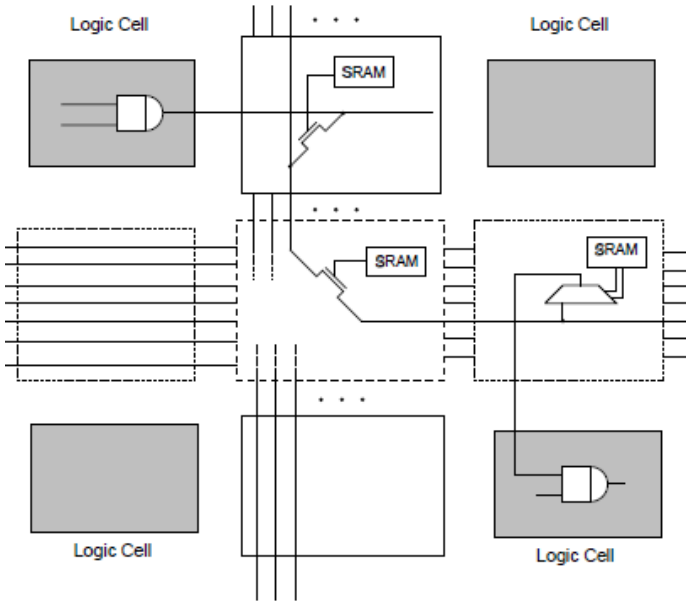


Figure 2.3: SRAM-Controlled Programmable Switch (Brown & Rose, 1996).

D. Programmable Routing: The routing architecture of an FPGA is determined by the programmable switches and the wiring segments that made the interconnections and how they are placed in order to realize the complete logic functionality of the FPGA device. Programmable routing comprises both long and short wire segment connecting configurable logic blocks, switch blocks and input/output blocks (Farooq *et al.*, 2012). The routing architecture can be very complex and composed of millions of pre-defined wires that can be configured by multiplexers and switches to build the desirable routing (Kastensmidt & Rech, 2015). The Wire segment is a programmable connection through a switch. One or more switches can be connected to the same wire segment. Each access point of a wire segment has at least one switch connected to itself (Battezzati *et al.*, 2011). The programmability is accomplished by turning on/off a switch controlling the connection of two or more wires (Habinc, 2002).

2.2.1.4 Configuration Memory

The configuration memory contains all the bits needed to program the circuit on the FPGA architecture. The configuration memory is also called bitstream. Each resource within the FPGA architecture is controlled by specific configuration memory bits. The bitstream is generally organized in small segments called frames and each frame is correspondent to a set of resources localized in a contiguous area of the FPGA architecture. These frames have a regular structure depending on the kind of resources and these frames are made of SRAM cells (Battezzati *et al.*, 2011).

These SRAM cells define the operation of the configurable logic blocks, routing resources, I/O blocks, and other programmable FPGA resources. The use of static SRAM memory

cells for configuration storage allows the device to be reprogrammed as often as necessary by reloading a new configuration memory (Wirthlin, 2015).

The configuration memory represents the largest portion of memory in SRAM-based FPGA device. This memory represents about 80% to 85% of the whole device memory. New high density SRAM-based FPGAs are composed of millions of SRAM cells. FPGA families fabricated in 28 and 20 nm technologies can have approximately one thousands of Megabits of configuration memory bits and a full reconfiguration can take up to 120 milliseconds (Jorge *et al.*, 2015). Figure 2.4 shows a general structure of an FPGA bitstream.

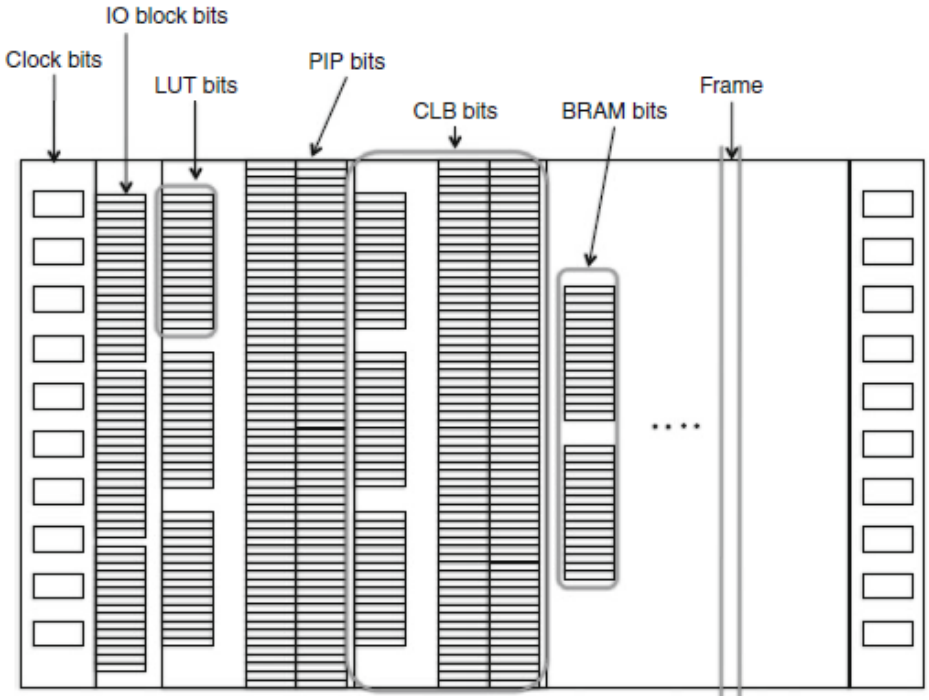


Figure 2.4: General Structure of an FPGA Bitstream (Battezzati *et al.*, 2011).

A. Configuration Memory Layout of Xilinx Virtex-5 FPGA: The configuration memory frame is the smallest addressable segment of the FPGA configuration memory and the frame size varies among FPGA families. In the case of Virtex-5 FPGA, it is composed

of 41 words of 32 bits (1,312 bits). Each frame has a unique address that describe the physical position in the FPGA floor plan, and the position in the floor plan is tie to a specific resource such as CLB, block random access memory (BRAM), IOB, Digital Signal Processing Block (DSPB), Clock bits (CLK) (Herrera-Alzu & Vallejo, 2011). Generally, the FPGA configuration memory floor plan is organized in rows and columns. Each column defines a specific type of resource and the rows divides each column in equally group of elements (Tonfat *et al.*, 2015). Figure 2.5 shows the configuration memory floor plan of Xilinx Virtex-5 FPGA.

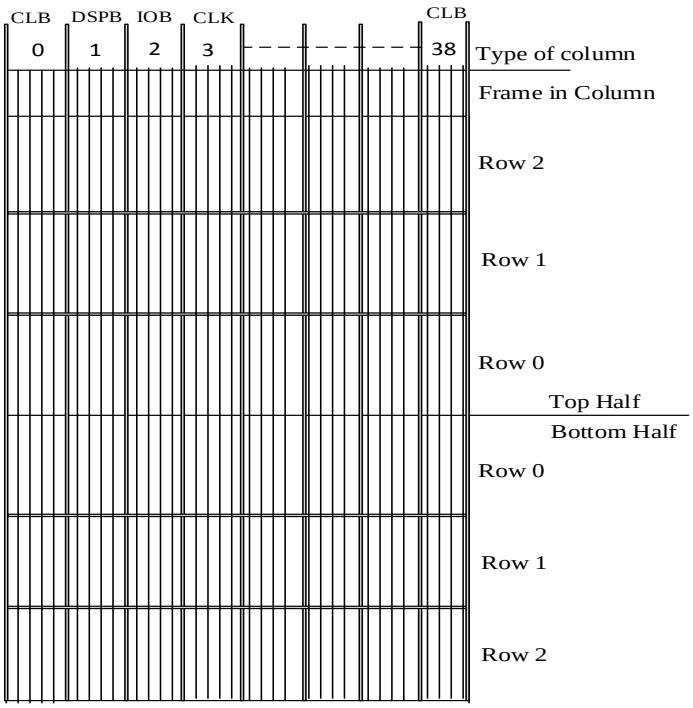


Figure 2.5: Configuration Memory Floor Plan of Virtex-5 FPGA (Tarrillo *et al.*, 2015).

Table 2.1 shows the types of resources and associated frame number in a typical Xilinx Virtex-5 FPGA.

Table 2.1: Number of Frames per Resources

Types of Resources	Associated Number of Frames Per Column
CLB	36
CLK	4
DSPB	28
Block RAM	30
IOB	54

Source: (Tonfat *et al.*, 2016).

B. Configuration Frame: FPGA device is organized into configuration frames which are the smallest addressable units and constitute the majority of SRAM cells in FPGA devices. The number and the size of the configuration frames vary from one device generation to another. About 82% of the total SRAM cells belong to the configuration frames (Rao *et al.*, 2014).

The number of frames in a bitstream depends on the size and type of the resources existing in the reconfigurable region (Tambara *et al.*, 2016). Table 2.2 shows the structure of a configuration bitstream.

Table 2.2: Configuration Bitstream Structure

Frame Header	Data Information
Command	Synchronization and other commands use for configuration operation
Frame data	Frames containing configuration bits
Command	Commands to stop configuration Process and desynchronize access port

Source: (Tambara *et al.*, 2016).

2.2.1.5 Static Random Access Memory Cell Memory Technology

The use of SRAM cell in FPGA is growing in critical based application areas such as space because of low application development cost, short time to market, and the reprogramming flexibility that they offer. Fundamental technology of FPGA consists of arrays of logic elements and registers, along with a configurable interconnection matrix. The current shrinking of device size helps FPGAs to be faster and denser, but at the same time also make them less reliable due to shrinking transistor size, the threshold voltage required to switch them, also shrinking makes transistors more prone to radiation induced errors (Tiwari & Tomko, 2005).

SRAM based FPGA will lose their configuration when power is removed, therefore, SRAM devices require a non-volatile configuration memory unit to store configuration data. This data must be loaded into the SRAM device after the FPGA is powered-on before use unlike the flash type FPGA whose cell is re-programmable but do not support partial reconfiguration though with non-volatile memory due to lack of vendor support for such capability (Codinachs & Weigand, 2009).

Flash technology is an Electrically Erasable Programmable Read Only Memory (EEPROM), the term flash comes from the erasure process of large blocks of data. The method of storing data is based on transistors with a "floating gate", which controls the behavior of the transistor. The floating gate is fully insulated and therefore holds a charge without a power supply. Writing and erasing are performed using a high voltage, which causes the charge in the floating gate to change through hot-electron injection and quantum tunneling (Miller *et al.*, 2007). The antifuse type FPGA which cannot be re-programmed in

other words they are one-time programmable devices with non-volatile memory (Ferguson & Tate, 2005).

SRAM technology holds the programming as the state of groups of cross coupled transistors are connected. Each group forms a bit of memory and determines if a switch is open or closed. When the supply power is provided, the transistors reinforce each other to hold the system in a state of "1" or "0". If the operating power is lost, the state of the transistor system is lost (Monmasson *et al.*, 2011).

The SRAM cell consist of a flip-flop otherwise known as latch. Flip-flop has two stable state, and on the storage nodes of a flip-flop, logical data "0" or "1" is stored, flip-flops represent the smallest portion of an internal state for SRAM FPGAs, flip-flops typically contain the most important state of the circuit and upsets in the flip-flops often cause significant disruption to the circuit's operation (Ishibashi & Osada, 2011).

A. SRAM Cell Architecture: The SRAM cell widely used is the full complementary metal oxide semiconductor (CMOS) 6-Transistor (6T) memory cell. The 6T static random access memory is a type of semiconductor memory that uses bi-stable latching circuitry to store each bit. The SRAM cell consists of two inverters P1, N1 and P2, N2 alongside with two access Metal Oxide Semiconductor Field Effect Transistor (MOSFET) N3 and N4 as shown in Figure 2.6 (Yang *et al.*, 2011).

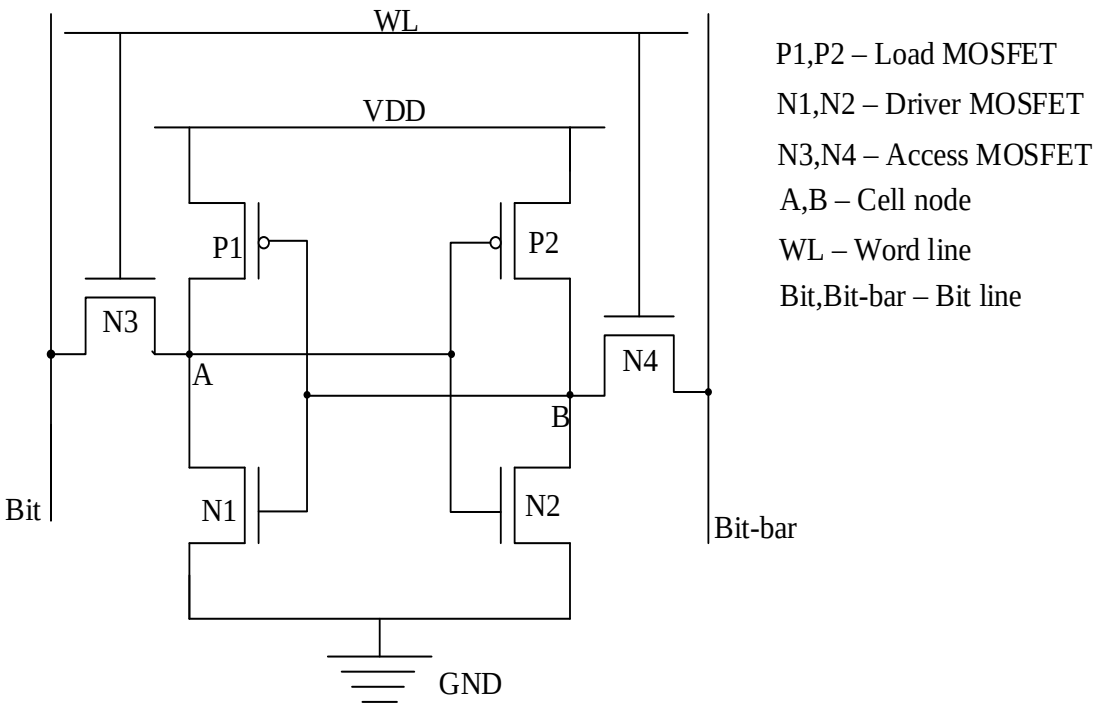


Figure 2.6: Schematic of a 6T SRAM Cell (Nigam *et al.*, 2015).

The two access MOSFET N3 and N4 are connected to a pair of bit lines bit and bit-bar. The two access MOSFETs are also connected to a Word Line (WL). To form a flip-flop, the input and the output of one inverter are connected to the output and the input of the other inverter, respectively (Nigam *et al.*, 2015).

B. Operational Principle of a 6T SRAM Cell: The SRAM cell operates in three different modes namely (Nigam *et al.*, 2015):

1. Standby mode (circuit is idle): In this mode, the word line is not triggered therefore the word line is 0V, so the pass transistors N3 and N4 which connect 6T cell from bit lines are turned off. It means that cell cannot be accessed. The two cross coupled inverters formed by N1-N2 will continue to feedback each other as long as they are connected to the supply, and data will hold in the latch.

2. **Read mode (data has been requested):** In this mode, the word line is triggered therefore the word line is “1”, so word line enables both the access transistors which will connect cell from the bit lines. Now values stored in node “A” or “B” are transferred to the bit lines to be accessed, either the value in node “A” or “B” is accessed. Assuming that “1” is stored at node A so bit line bar will discharge through the driver transistor N1 and the bit line will be pull up through the load transistor P1 toward VDD, hence a logic “1” is confirmed. SRAM cell technology ensures read stability therefore data are not disrupted during read operation.
3. **Write mode (updating the content):** If “1” is initially stored in the cell and needs to be written with a “0”, the bit line is lowered to 0V and bit bar is raised to VDD, a cell is selected by raising the word line to VDD, for SRAM to operate in write mode it must have write-ability which is minimum bit line voltage required to flip the state of the cell.

The power consumption of SRAM varies widely depending on how frequently it is accessed, it can be high when used at high frequencies. On the other hand, SRAM used at a somewhat slower pace at lower frequency draws very little power and can have a nearly negligible power consumption when sitting idle in the region of a few micro-watts (Aishwarya & Mahendran, 2016).

2.2.2 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. SEU 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 (Jacobs *et al.*, 2012)), this charge transfer can change the voltage level of critical nodes within the configuration memory cell of an FPGA such that the modified voltage level reflects the opposite state of the cell (by 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 (Wirthlin, 2015).

Generally, when an energetic particle strikes one of the sensitive (critical) nodes of an SRAM cell in the configuration memory of an FPGA, the effect can produce an inversion in the stored value, in other words, a bit-flip in the memory cell occurs (Kastensmidt & Carro, 2006).

SEU is a bit-flip that occurs when the ionizing particle hitting a transistor of a memory cell deposits enough charge to revert the state of the cell. The memory cell still works perfectly in the sense that a write or read operation is performed normally that is why SEU are also called soft errors (or temporary fault) as the SRAM cell state of the FPGA is not damaged, but the stored information is corrupted, as a result by rewriting the correct value a recovery is performed. When the ionizing particle hits a logic cell it generates a voltage spike which if latched can lead to a single event transient (SET) (Asadi *et al.*, 2007). SETs causes unwanted glitches that propagate through combinational circuitry. Transient errors do not alter SRAM configuration bits, but if the temporary glitch is latched into a sequential circuit (flip-flop or latch), it will appear as a single event upset. Again, the logic cell is not damaged in the sense that a new operation will eventually be correctly performed. Soft errors can be transient and stochastic making them to be quite difficult to be identified and

corrected (Kastensmidt & Rech, 2015). Virtex FPGAs offer the possibility to read the configuration memory at any time.

Readback is a process where the configuration memory is continuously read or scanned during which SEUs are detected in the configuration memory of SRAM-based FPGA (Bolchini *et al.*, 2011).

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 CLB or 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.

FPGAs are also sensitive to SEUs in BRAM and user flip-flops. BRAMs are often used as memories in FPGA designs. User flip-flops are the registers in the FPGA that are represented in a design for use in state machines, counters, and other sequential logic structures. SEU in the configuration memory bits of an SRAM-based FPGA has a persistent effect and it can only be corrected when a new bitstream is loaded to the FPGA (Johnson & Wirthlin, 2009).

Figures 2.7 - 2.10 show how a change to a configuration memory bit can affect the function of a LUT or the routing between nodes, and cause failure in the user design (Harward *et al.*, 2016).

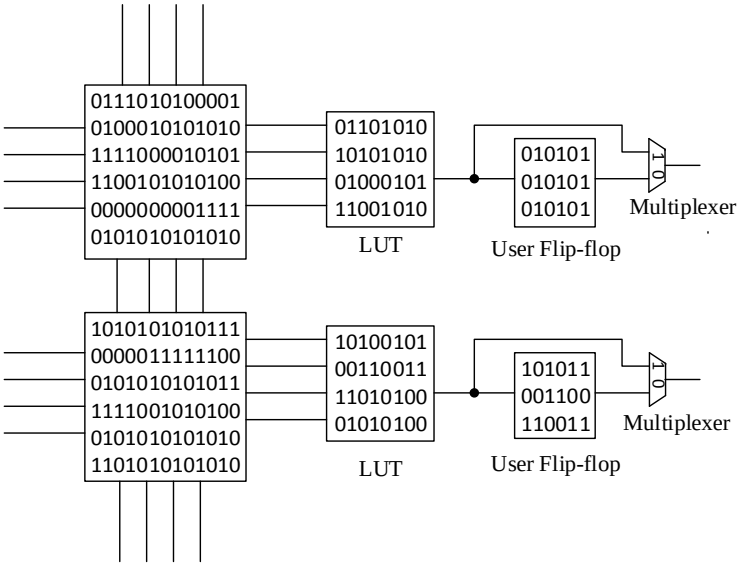


Figure 2.7: Illustration of Configuration Memory (Harward *et al.*, 2016).

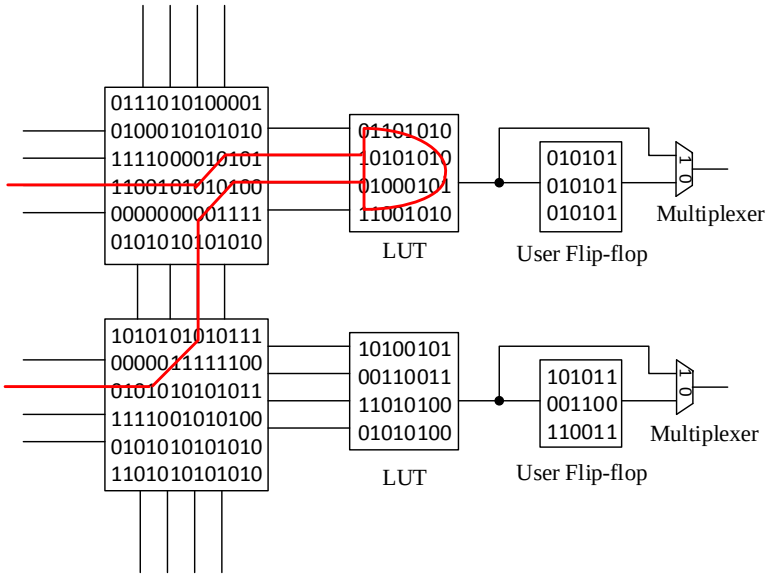


Figure 2.8: Configuration Memory used to specify Logic and Routing (Harward *et al.*, 2016).

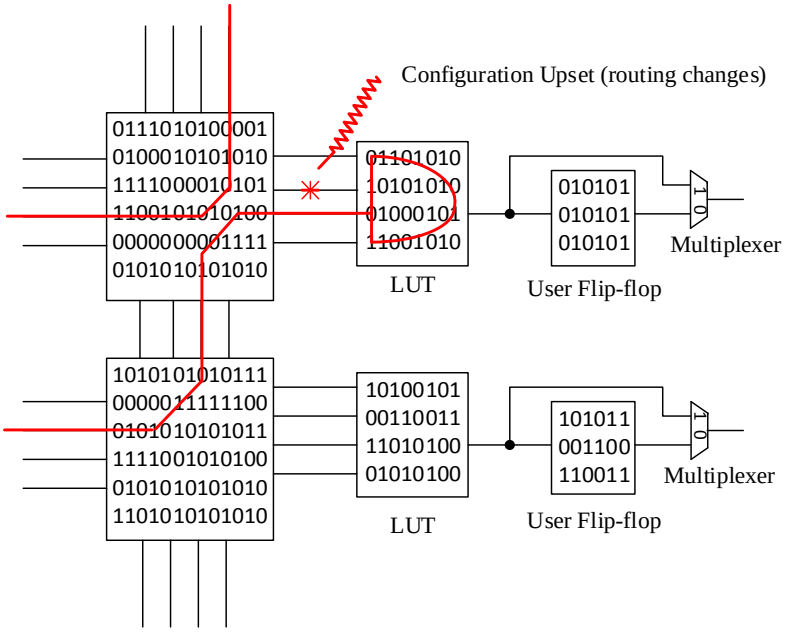


Figure 2.9: Upset in Routing (Harward *et al.*, 2016).

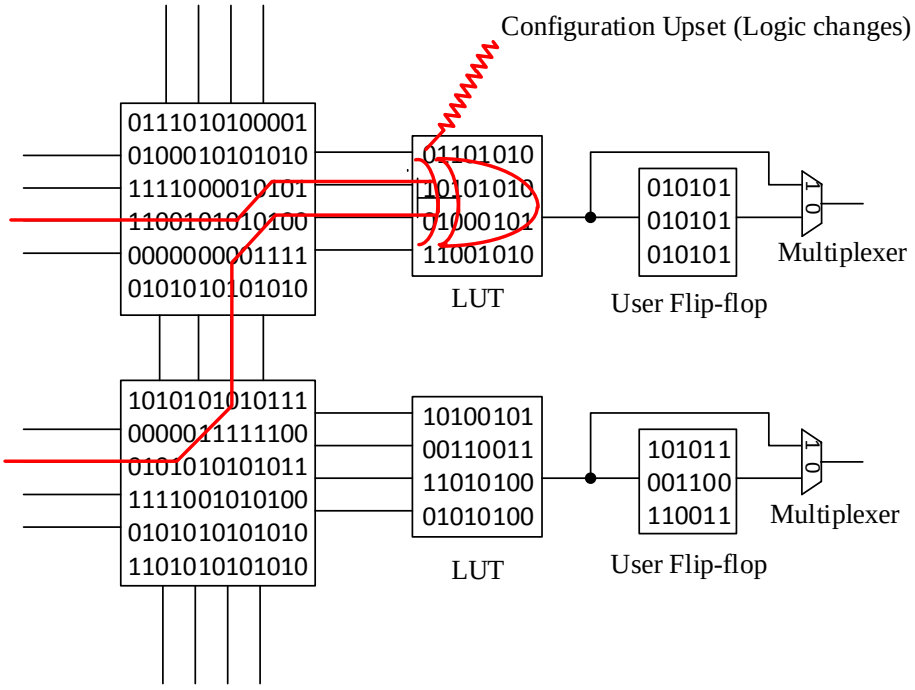


Figure 2.10: Upset in Logic (Harward *et al.*, 2016).

SEUs within the configuration memory may alter the function of the configurable logic blocks, the routing network, or modify the operation of the I/O blocks or any other logic

resource. Figure 2.8 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 2.9 that disconnects one input of the ‘‘AND’’ gate. The second configuration upset as depicted in Figure 2.10 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. And if such errors keeps on occurring, this leads to functional failure of the FPGA (Harward *et al.*, 2016). Not all upsets to the configuration memory will cause the design to deviate from its intended function. Some of the configuration bits associated with logic and routing are unused by the circuit and, thus, upsets in these bits have no effect on the design. Configuration bits that cause a design to fail when upset occur are called ‘‘sensitive or critical’’ while that do not impact the design are called ‘‘insensitive.’’ The configuration memory on an FPGA can be protected from SEUs with memory scrubbing and/or Error Detection and Correction (EDAC) techniques. FPGA fault-tolerant design techniques such as TMR can also be employed against SEUs (Wirthlin, 2015).

Error in the bitstream may be as a result of SEU when a radiation strike causes only one configuration memory cell to change its logic state, or as a Multiple Bit Upset (MBU) when a single high-energy charged particle forces several adjacent configuration bits to flip or when the upset cells belongs to the same frame for an FPGA. (Johnson & Wirthlin, 2009). SEU in configuration layer may propagate as a functional failure in the application layer, independently of the user functionality implemented. SEU in a routing resource is

very critical even if not used by the application layer, may also have functional effect and changing the behavior of the application as they may create shorts or bridges between logic resources. Indeed, soft errors in routing resources induces single or multiple errors in the application layer. The situation is dramatic considering that more than 60% of the configuration bits in modern Virtex FPGAs are linked to routing resources, although 10 to 20% of them are typically used. On the other hand, configuration memory of SRAM-based FPGA density keeps on growing in every new FPGA generation, thus increasing the MBU due to smaller geometries of transistors which are smaller than ionized particle track size, and lower operating voltage (Badodekar, 2009). For this reason, SEU is a well-known threat for space systems directly exposed to cosmic rays and solar flares and this has to be meticulously addressed in order to use FPGAs reliably in hostile space and avionics application environments (Lanuzza *et al.*, 2010). Thus, hardening the design mapped on the Virtex FPGA is a considerable option against SEU. Design hardening such as TMR is one of the techniques employed to mitigate SEUs. Hardening by design include introducing hardware and/or software redundancy (Samudrala *et al.*, 2004).

2.2.3 Total Ionization Dose

The use of an FPGA in radiation environments such as space may result in the occurrence of faults, the release of charge from high energy particles can induce couples of electrons-holes in the device, such that current spikes are generated, the accumulation and impact of heavy particles can also cause modifications in the electrical characteristics of the semiconductor material lattice. This effect is commonly known as the Total Ionizing Dose (TID) and can cause permanent faults to appear in a circuit (Ottavi *et al.*, 2008). Ultimately, radiation will cause functional failures within the device. Therefore, the

amount of radiation dose that a device can tolerate before failing to meet expected parameter specifications is called total ionizing dose (Virtanen, 2013).

The creation of electron-hole pair within the gate-oxide of CMOS-based devices such as FPGAs due to ionization radiation changes the threshold voltage of transistors and degrades the mobility of routing resources. These effects result in slower switching speeds and reduced drive capability (Wahle *et al.*, 1990).

Cumulative effects are gradual effects taking place during the whole lifetime of the electronics exposed in a radiation environment (Faccio, 2001). The creation of electron-hole pair occurs when an electron in the valence band is excited across the band gap into the material's conduction band. The excited electron thus also leaves a hole behind in the valence band, and if electric field is present, the electrons are readily swept away because their mobility in silicon is much greater than that of the holes. Except for some small fraction of pairs that undergoes recombination immediately, the created electrons and holes are free to drift and diffuse in the material until they undergo recombination or are trapped (Maurer *et al.*, 2008). In other words, the electron-hole pairs created in this material do not completely recombine in a very short time. In the presence of an electric field in the oxide, a great amount of the pairs does not recombine, and both electrons and holes start to drift in the electric field. Electrons with a much higher mobility, can easily leave the oxide, while holes instead can be trapped in the oxide. In addition, this process can create (or better activate) defects at the silicon-oxide interface. The charge buildup and the activation of defects are the two reasons for device degradation induced by TID (Faccio, 2001).

Figure 2.11 shows the effect of a high neutron particle strike on the depletion region of a semiconductor device.

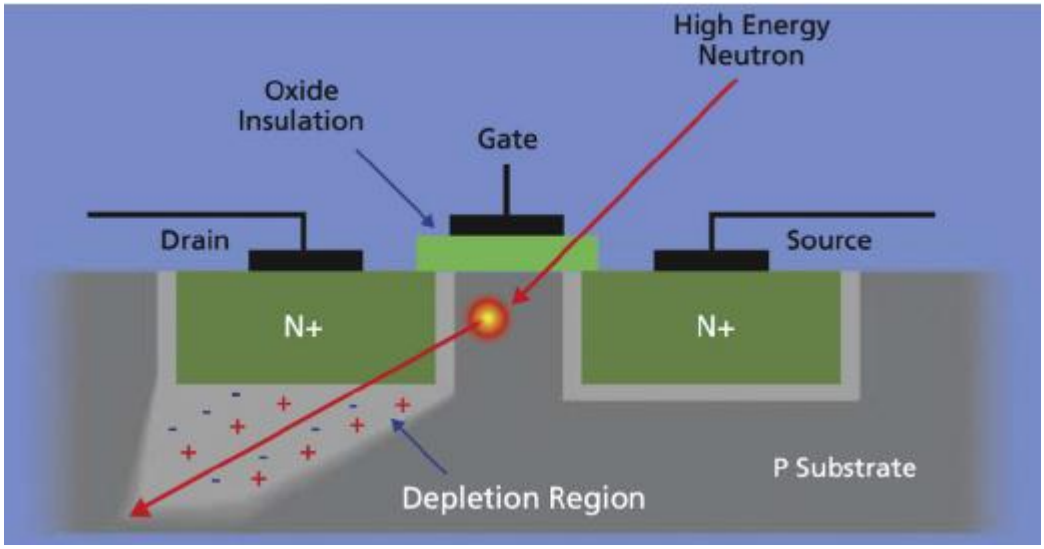


Figure 2.11: Effects of Neutrons on Semiconductor Device (Ciani & Catelani, 2014).

2.2.3.1 Effect of Total Ionization Dose

Total ionization dose causes three types of effects namely (Ranta, 2012):

A. **Performance:** This effect leads to having slower devices, whose maximum operating frequency is reduced, technology scaling is proportional to increase in the absorbed dose.
B. **Degradation:** This leads to higher leakage currents that increase power consumption when transistors are not used.
C. **Power consumption increase and programmability loss:** This leads to losing re-programmability in the FPGA configuration memory. Threshold voltages of memory elements, transistors shifts with the increase in the absorbed dose causing such a misbehaviour.

TID effects are severe faults that cannot be removed by means of a simple power cycle, because they modify the lattice structure of the materials inside the FPGA (Battezzati *et al.*, 2011).

2.2.4 Radiation Environment

Radiation can be said to be set of particles, charged or not, that can interact with the electronic system by an exchange of energy. Considering the broad range of applications of electronic systems, the most harshly environment from the radiation point of view is space (Ozge, 2010). The Earth's magnetic field acts as a partial shield against charged particles which come from inside and outside the Solar System. Electronic circuits used in terrestrial applications are consequently safer than the ones used in space or in avionic applications (Battezzati *et al.*, 2011). The use of FPGA in radiation environment within spacecraft is motivated by the growing computational needs associated with modern sensors used in spacecraft. Because of the tremendous amount of data generated by modern sensors, it is no longer possible to send all sensor data back to earth for processing. Today, much of that processing (including data compression) must be done on the spacecraft. FPGA provides an attractive solution for many of the computationally intensive on-board processing tasks used by modern spacecraft. Modern FPGAs contain a tremendous amount of computational resources due to the large number of logic elements, internal memory, and dedicated Digital Signal Processors (DSPs) (Wirthlin, 2015). The radiation environment is composed of various particles generated by sun and stars activity. The space is full of galactic cosmic rays, which are heavy ions produced by explosion of supernovas (a star that explodes and becomes extremely luminous in the process), the atoms released in the process losses protons or electrons and thus, gain charge which are accelerated reaching energies in the order of Giga electron-Volt (GeV) (Kastensmidt & Rech, 2015).

There are three main environments of concern for space electronics namely (Ozge, 2010):

A. **Van Allen trapped radiation belts:** The earth's magnetic field is responsible for trapping high energetic particles within the radiation belt referred to as Van Allan belts. This is a radiation belt where highly energetic protons and electrons are held in place by the earth's magnetic field. It is believed that the protons and electrons of the inner belt originate from the decay of neutrons produced when high-energy cosmic rays from outside the solar system collide with atoms and molecules of the earth's atmosphere. Due to the off-center dipolar field of the earth, the Van Allan belts reaches their lowest altitude in what is referred as the South Atlantic Anomaly (SAA). In other words, SAA is the region where the inner van Allen radiation belt makes its closest approach to the planet earth's surface. For systems in the Low Earth Orbit (LEO), spacecraft passing through the SAA are exposed to an increased level of radiation which affects electronic devices especially SRAM-based FPGAs. The outer Van Allen belt contains charged particles of both atmospheric and solar origin, the latter consisting largely of helium ions from the solar wind.

B. **Galactic cosmic rays:** This consists of protons and heavy ions with very high energies such as neutrons. Neutrons are created also by cosmic ion interactions with the oxygen and nitrogen in the upper atmosphere, neutrons are particularly troublesome, because they can penetrate most manmade constructions. Galactic cosmic rays are so energetic to pass the Van Allan belts and hit the upper level of the terrestrial atmosphere.

C. **Solar flare particles:** Solar flare occurs when magnetic energy that has built up in the solar atmosphere is suddenly released. These flares are sudden flash of brightness observed near the sun's surface. It involves a very broad spectrum of emissions

making the radiation emitted covers the entire electromagnetic spectrum ranging from radio wave with longer wavelength through gamma rays with shorter wavelength. Energy release ranges to 1×10^{20} joules for a typically well observed solar fare event. Solar fare particles comprises of protons and a small percent of heavy ions. These types of particles exhibit low energy. It is worth noting that solar wind are trapped in the Van Allen belts due to its low energy.

2.2.5 FPGA Single Event Upset Scrubbing Techniques

In the recent past, work has been carried out on SRAM-based FPGAs with soft errors mitigation properties tackling faults hitting the device configuration memory in radiation environment. EDAC techniques are typically used in user logic and memories for preserving design functionality and data integrity in the event of SEUs. Dynamic (without interrupting the application layer) re-configurability of the configuration memory can be used, not only for reconfiguring or upgrading the application, but also for mitigating soft errors before they accumulate and totally alter the circuit design in the Application layer (Kastensmidt & Rech, 2015).

Configuration memory scrubbing is the process of repairing upsets within the configuration memory by writing the correct configuration data back into the configuration memory.

A more complex but efficient configuration scrubbing process involves periodically or continuous readback process of the configuration memory for error detection and correction. Scrubbing occurs while the FPGA is powered-on and operational in order to mitigate against SEU. Therefore, a scrubber is a circuitry that basically access the configuration memory for error detection and correction. Scrubbers comes with a variety

of trade-off including size, power consumption, performance, robustness etc. (Wirthlin & Harding, 2016).

Configuration scrubbing preserves the integrity of the memory. Scrubbing involves a continuous process of reading the current state of the configuration memory known as readback process and writing correct data back into the memory (Heiner *et al.*, 2008).

Configuration memory scrubbing against SEU and MBUs can be detected by reading back the FPGA configuration memory via the configuration port, and comparing it with a golden reference stored in a non-volatile memory (golden memory). After the error is detected, scrubbing is performed, and the minimum reconfigurable element in the configuration memory is the frame.

Configuration scrubbing is an essential component of any high-reliable FPGA based system (Johnson & Wirthlin, 2010). Increasing the scrubbing rate improves the system reliability (Ibrahim *et al.*, 2013).

2.2.5.1 Scrubbing Methodologies and Architectures

Scrubbing Methodology and Architecture against SEUs in the configuration memory of SRAM-based FPGAs have a persistent effect and they remain until the original configuration is rewritten. Choosing the right methodology for an application typically implies trading-off reliability, complexity, power consumption, cost, scrubbing time and area overhead (Jorge *et al.*, 2015). Figure 2.12 shows different scrubbing classifications.

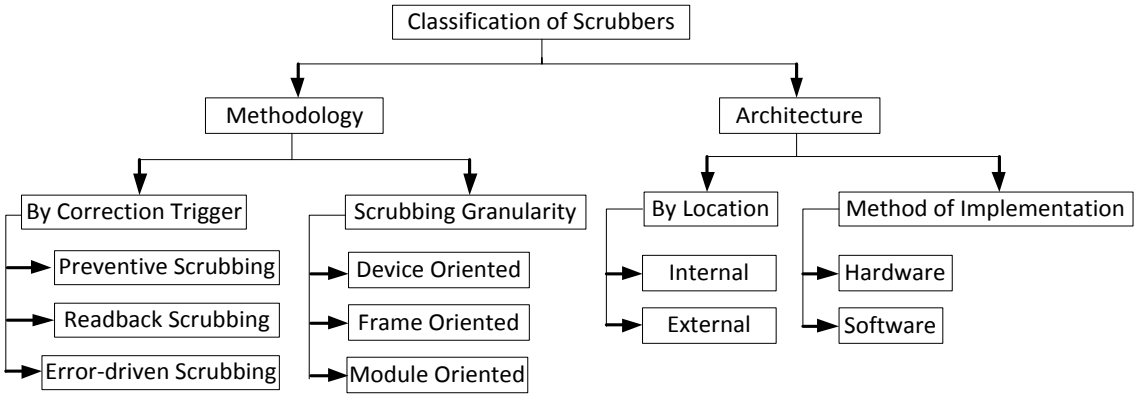


Figure 2.12: Scrubbers Classification (Jorge *et al.*, 2015).

The scrubbing methodology can either be by correction triggered or scrubbing granularity. The first methodology is the preventive scrubbing also known as a blind scrubbing where the scrubber continuously rewrites the configuration memory after a predefined scrub interval regardless of whether an error occurs or not, this is the simplest methodology. On the other hand, readback scrubbing refers to periodically reading the configuration memory until a soft error is detected; only then a scrub cycle is executed. With a readback scrubbing methodology, it is possible to achieve lower energy consumption since only when a soft error is detected, the scrubbing is enabled. In the third methodology, a scrub cycle is enabled when the scrubber receives an error signal from the functional design. This can be used in TMR designs where the voter can detect a module with functional error (Jorge *et al.*, 2015).

In the device oriented scrubbing granularity, the scrubbing process covers the FPGA bitstream as a whole. This scrub method is simple and has a minimum processing overhead. While in the case of frame oriented scrubbing, the configuration memory is typically scrubbed on a frame-by-frame basis. Frame-based scrubbing requires a more complex scrubber implementation because each frame must be prepared. Module

scrubbing refers to certain portion of the configuration memory to be scrubbed (Siegle *et al.*, 2015).

Scrubbing can be implemented using an internal or external interface. When external interface is used, the scrubber is implemented outside the FPGA in a radiation-hardened device such as anti-fused based FPGA or an ASIC, this ensures that the scrubber logic itself is not affected by SEU. The scrubber uses the selectable Microprocessor Access Port (SelectMAP) interface which has high data throughput to access the bitstream. On the other hand, when the scrubber is implemented inside the FPGA, and the Internal Configuration Access Port (ICAP) interface is used, which is a replica of the SelectMAP interface. Internally implemented scrubber logic are susceptible to radiation thereby, making them less reliable than the external ones. However, fault tolerant scrubbers improves the device reliability by applying TMR to the scrubber. The use of ICAP removes the need for external circuit components, additional input-output pins and radiation hardened configuration memories (Heiner *et al.*, 2008).

When scrubbing is performed internal to the FPGA, this allows scrubbing to be performed very fast because no addition of specialized external circuitry is incorporated to the scrubber. An external scrubber runs much slower than the scrubber within the FPGA (Harding & Wirthlin, 2014).

2.2.5.2 Scrubbing Mechanism

To perform FPGA scrubbing, the configuration data is typically read in sequential order from start to finish. The configuration memory is read, and the scrubbing system compares this data against a golden data stored in an external radiation memory (in the case of

duplication and compare technique), if a discrepancy is found between the configuration data within the device and the golden configuration data, the scrubbing system will repair the corrupted data by writing the correct golden data into the FPGA. If there is no discrepancy between the configuration data and the golden data, the scrubber moves on to the next set of data in the next frame. Once the scrubber has reached the end of the configuration memory, the process is repeated again from the beginning. This process is illustrated in Figure 2.13 (Wirthlin & Harding, 2016).

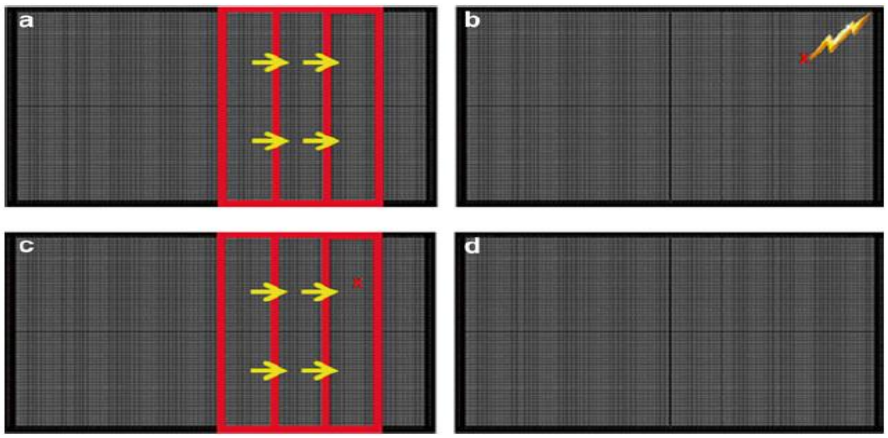


Figure 2.13: Scrubbing Mechanism (Wirthlin & Harding, 2016).

The scrubbing mechanism as shown in Figure 2.13 describes the processes involved in mitigating SEU as a result of radiation strike in the FPGA configuration memory, explained in the steps below:

Figure 2.13a shows an FPGA configuration memory read frame by frame progressing from left to right.

Figure 2.13b shows radiation strike causing an upset in the FPGA configuration memory.

Figure 2.13c shows a scrubber progressing through the FPGA bitstream and eventually finds the frame with the error and corrects it.

Figure 2.13d shows the scrubbed frame and the SEU mitigated (Wirthlin & Harding, 2016).

Configuration scrubbing is not sufficient to prevent configuration upsets from temporarily changing the behavior of an FPGA circuit. Even with high-speed configuration scrubbing, there is a period of time between the occurrence of an upset in the configuration memory and the repair of the upset through scrubbing. During this time period, the circuit may be operating incorrectly, entering incorrect states, or producing incorrect results (Wirthlin, 2015). For reliability, scrubbing needs to be coupled with a redundancy technique such as TMR to mask the soft error while waiting for reconfiguration (Johnson & Wirthlin, 2009).

2.2.6 Triple Modular Redundancy

Faults are defined as any bit-flip known as SEU in the configuration memory detected during readback process. Fault tolerance becomes a substantial design criteria for the applications where the reliability of a device is crucial. Fault tolerance is defined as a technique that provides a service capable of fulfilling the system function in spite of limited number of faults, fault tolerant techniques has increased in Integrated Circuits (ICs) such as FPGA operating in hostile environment such as space (Kastensmidt & Rech, 2015). Spatial redundancy is based on the replication of n-times the original module building n-identical redundant modules, where outputs are merged into a majority voter. Usually “n” is an odd number, the voter decides the correct output by choosing the majority of the equal output values. The most common case of n-Modular Redundancy (nMR) is when n is equal to three, where it is called TMR. In this case, a majority voter is used that is able to vote out two out of three values that are fault free (Tarrillo & Kastensmidt, 2016).

TMR is the most applied fault masking technique for fault tolerance of software or hardware systems (Alagoz, 2009). TMR is one of the approaches used in mitigating the effects of SEUs, TMR provides protection against all single point failures. With this approach, the user design is triplicated and the outputs from each module are fed into a voting circuit to determine the correct output value, TMR results overhead in terms of area and power consumption (Cetin & Diessel, 2012). Figure 2.14 shows an implementation of TMR where a fault in any of the redundant hardware modules will not produce an error at the output as the majority voter will select the correct result from the two working modules achieving fault masking properties (Johnson & Wirthlin, 2010).

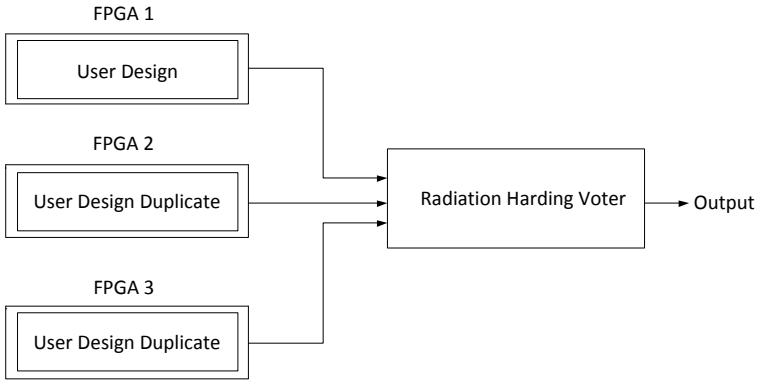


Figure 2.14: Implementation on an FPGA TMR (Codinachs & Weigand, 2009).

There are two types of TMR variations namely (Miller *et al.*, 2007):

A. External: This type of TMR uses three independent FPGAs that run the same set of operations at the same time in parallel where each FPGA implements a module replica and the outputs are connected to an external radiation-hardened voter that compares the results. External TMR requires significant hardware overhead, each protected module is triplicated and board layout complexity is significantly increased, but is reliable.

B. Internal: This creates three identical modules within a single FPGA, and the majority voter resides internally. Internal TMR can reduce the number of physical FPGAs, area overhead and power consumption required for critical application.

Configuration scrubbing prevents error accumulation in FPGA configuration memory. While TMR masks individual errors, it does not correct the underlying fault and cannot correct errors that occur in multiple modules, this is the traditional use of TMR (Miller *et al.*, 2007). TMR and configuration scrubbing are very effective at mitigating against the effects of configuration SEUs, these techniques cannot prevent the possibility that multiple configuration upsets will occur within a single scrub cycle as it is the norm (Bolchini *et al.*, 2011). TMR with triplicated voters is an effective reliability technique for protecting configuration memory compared to other methods. Although TMR can effectively reduce malfunctioning in the configuration memory but cannot correct errors but only masks the errors (Li, 2012).

The probability of an error occurring at the same time in two modules is significantly reduced (Nazar & Carro, 2012). Therefore, considering technological trend, SRAM-based FPGAs have more and more probability of having MBU, hence the use of TMR is an attractive solution at design level on systems integrated into a single FPGA (Tarrillo & Kastensmidt, 2016).

2.2.7 Frame Level Redundancy Scrubbing

Scrubbing is a technique employed to correct upset in SRAM-based FPGA configuration memory. FLR scrubbing technique offers better performance when compared to the classical blind scrubbing. FLR scrubbing technique relies on the generation of TMR design with the certainty that all the three modules have exactly the same data in the configuration

memory. Any frame with an upset(s) can be corrected using the data from the other two modules in the TMR network by performing a bit level majority voting. MBU and SEU are corrected using FLR scrubbing.

The premise is that the probability a frame in two or three of the TMR module with the same frame address will have upset at exactly the same cell position within the same scrub cycle is extremely low due to the incredibly vast logic bits in the FPGA configuration memory. But even if by chance a wrong logic bit is mistaken to be a correct logic bit, the upset will propagate to the next frame. This type of error will remain until the next full reconfiguration of the device (Tonfat *et al.,* 2015). Figure 2.15 shows the FLR scrubbing flow chart.

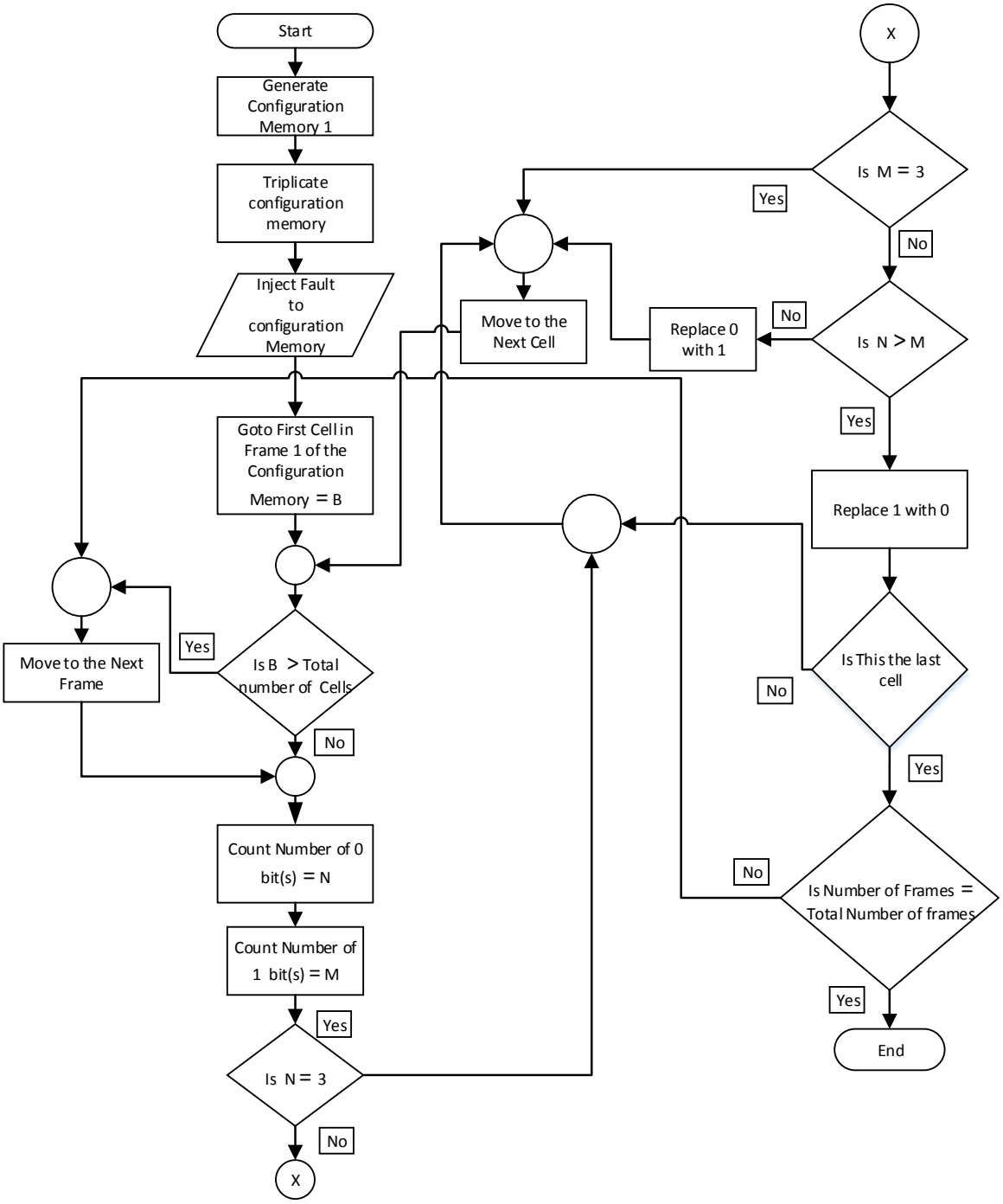


Figure 2.15: Frame Level Redundancy Scrubbing Flow Chart (Tonfat *et al.*, 2015).

2.2.8 Cyclic Redundancy Check

Memory devices with very high data requires very fast supporting systems for computation. 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 and Cyclic Redundancy Check (CRC) is reliable for such application. CRC is highly efficient and involves computational process. (Akagic & Amano, 2012). CRC is an error detection scheme that detects corruption of digital content during data transmission, processing and in memory storage devices. CRC is a good option for error detection scheme used since it has very low probability of undetected errors (Akagić & Amano, 2011). CRC as an error detection mechanism 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 (Battezzati *et al.*, 2011).

The CRC mechanism requires k-bits code words and r-bits redundancies as its suffix. Before the encoding a fixed generator polynomial $g(x)$ is selected. The encoding rule of CRC is according to k-bits information words code to compute the r-bits redundancies. The k-bits of storage data can be expressed as coefficient of polynomial. The word codes can be expressed in Equation (2.1) (Zhang & Ding, 2011):

$$M(x) = a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \ldots + a_1 x + a_0 \tag{2.1}$$

$$G(x) = x^3 + x^2 + 1 \tag{2.2}$$

where:

$M(x)$ is the data in the configuration memory

$G(x)$ is the generator polynomial

a_n is coefficients of the polynomial

x is the position of the code

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 redundancies. The following steps are executed (Zhang & Ding, 2011):

A. Generating a multinomial by multiplying X^{n-k} with $M(x)$ to give Equation (2.3).

$$x^{n-k}M(x) \tag{2.3}$$

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 Equation (2.4).

$$\frac{X^{n-k}M(x)}{G(x)} = Q(x) + \frac{R(x)}{G(x)} \tag{2.4}$$

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 shown in Equation (2.5)

$$X^{n-k}M(x) = G(x)Q(x) + R(x) \tag{2.5}$$

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 (Akagić & Amano, 2011). The configuration bitstream is stored inside the FPGA in the form of frames in the FPGA SRAM, CRC can be calculated for each frame for error detection during readback process (Ibrahim *et al.*, 2013). Depending on the size or number of bits in each frame of an FPGA one can include just enough redundant bits to make it possible to detect errors in frames with upset. A 32-bit CRC is able to detect any change in as many as 2^{32} bits (Chapman & Jones, 2010).

2.2.8.1 Requirements of Cyclic Redundancy Check

In order to design a valid CRC value or code, the following major requirements are to be satisfied:

A. The CRC should have exactly one less bit than the generator polynomial G(x).

B. Appending the CRC to the end of the data unit ($x^{n-k}M(x)$) should result in the bit sequence which is exactly divisible by the divisor $G(x)$.

It should be noted that the redundant bits appended to the message word M(x) to yield $x^{n-k}M(x)$ before dividing with the fixed generator polynomial G(x) in an Exclusive-OR operation is one bit less than the $G(x)$.

2.2.9 Single Event Upset Detection and Correction Time

Configuration scrubbing is an essential component of any high reliable FPGA based system. Configuration scrubbing prevents the buildup of multiple configuration upsets by detecting upsets and correcting them as soon as they are found. Scrubbing limits upsets in the configuration bitstream which is dependent on the scrub rate (Heiner *et al.*, 2008).

The worst case time to correct error in an FPGA configuration memory via scrubbing process can be in the order of milliseconds for critical applications with hard real-time constraints. Furthermore, scrubbing is unable to deal with faults sufficiently meaning that, if a more comprehensive fault tolerance mechanism is to be provided, other techniques have to be applied in conjunction with TMR (Nazar & Carro, 2012).

The time required to correct an error is associated with the time required to traverse the configuration memory. It is usually in the order of milliseconds as mentioned earlier and gets longer as more resources are embedded into FPGAs, which increases the size of configuration memories, and if the system is unavailable during the time required to remove the error, several real-time deadlines may be missed, which can be catastrophic for critical application systems (Nazar *et al.*, 2013).

Maintaining the integrity of an FPGA configuration memory is therefore crucial to ensure the reliability of the system, due to its large size compared to other device structures, FPGA configuration memory faults pose a major dependability threat. Moreover, the size of the configuration memory also directly affects the time required to repair it (Nazar, 2015). The configuration memory scrubbing has a power overhead, it is well known that the power consumption depends on the readback, scrub rate and the scrub methodology adopted (Jorge *et al.*, 2015).

2.2.10 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 (Tarrillo & Kastensmidt, 2016).

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. SRAM-based FPGA devices are composed of fix number of transistors in their SRAM cell and the arrangement of logical blocks, configurable interconnects and special blocks such as RAMs and DSP modules and other resources are the cause of extra static power consumption (Tarrillo & Lima, 2014).

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 by equation (2.6), (2.7).and (2.8) respectively (Tarrillo & Kastensmidt, 2016):

$$P_T = P_D + P_S \tag{2.6}$$

$$P_S = V_{CC} + I_{CC} \tag{2.7}$$

where:

V_{CC} is the voltage level

I_{CC} is the leakage current

$$P_D = \sum_{i=1}^{n} \beta_i C_i f V_{CC}^2 \tag{2.8}$$

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 (Tarrillo & Kastensmidt, 2016).

When the scrubber is internal to the FPGA, both components add up and contribute to the overall FPGA power consumption and dissipation. When the scrubber is an external device, only considering its static power dissipation makes the overall system power increase substantially. Since scrubbing process is tied to power consumption, performing readback and scrubbing operation at faster rate minimizes the error correction time and improves the Configuration memory reliability (Herrera-Alzu & Lopez, 2013).

2.2.11 Fault Injection

Fault injection involves intentionally inserting corrupt data into the configuration memory artificially by partially configuring FPGA frames within the configuration data in which one or more configuration bits are flipped (a logic “0” becomes logic “1” and vice versa), faults are usually injected one at a time. Bit flip emulates a SEU in the configuration memory cells of an SRAM-based FPGA (Wirthlin *et al.*, 2016). With the information about the organization of the configuration memory and the specific commands sequence to read and write frames, we can flip any bit of the configuration memory thus emulating the effect of an SEU (Tonfat *et al.*, 2016). Artificial injection of faults into the

configuration memory can be used to verify the effectiveness of TMR and related techniques (Wirthlin, 2015). Fault injection helps to understand the benefit of a mitigating technique (Wirthlin *et al.*, 2016).

2.2.11.1 Category of Fault Injection Techniques

These are classified as Physical fault injection (which includes hardware and software-implemented fault injection techniques), Simulation-based fault injection technique and FPGA-based fault injection technique.

A. Hardware and Software-based Fault Injection: These are classified under physical fault injection technique when a prototype of the system is already available, or when the system itself is too large to be modeled and simulated at an acceptable cost. In physical fault injection techniques, a manufactured prototype of the circuit or design is exposed to radiations and this produces realistic results but they need very expensive facilities. The software based-injection technique just consists of reproducing at software level the errors that would have been produced upon occurring faults in the hardware.

B. Simulation-based Fault Injection: This is very effective in allowing detailed analysis of a designed systems, since it can be exploited when a prototype is not yet available, and allows the analysis of practically any possible fault. The simulation based fault injection technique is categorized into two main groups:

1. **Reconfiguration based technique:** This technique changes the configuration memory bits of the FPGA device using full or partial reconfiguration in such a way that a target fault model is applied on the desired fault site, in other words fault in the form of SEU is injected in the desired location in the configuration memory.

2. **Instrumentation based technique:** In instrumentation-based technique fault injector circuits called saboteurs are added to each location where fault or upset is desired in the configuration memory. Saboteurs are only added to the design during fault injection campaign or they are added during the design and remains inactive during the normal operation of the FPGA, but when they are activated the inject faults during the fault injection process.

C. FPGA-based Fault Injection: This method which is also known as Emulation based Fault Injection technique was employed because of FPGA properties such as re-configurability, higher speed, higher performance and cost effectiveness. The Emulation based Fault Injection technique combines the advantages of Simulation and Physical techniques, which are controllability and increase speed of the fault injection process (Ebrahimi *et al.*, 2014).

2.3 Review of Similar Works

SEU in SRAM-based FPGA has been an active research topic and different techniques have been presented for solving this problem. Most of the techniques employ involves a trade-off which is dependent on the sensitivity in the area of application being used. The review of similar work are presented as follows:

Carmichael & Brinkley Jr., (2006) presented a novel technique for mitigating SEU in SRAM based FPGA. The technique involved the use of redundancy such as TMR were three identical FPGAs and a voting circuit where implemented inside or outside the FPGA to determine the correct output or only sensitive logic resources were triplicated. An error detection technique whereby the configuration memory was readback and a CRC code

which was also referred to as checksum was generated for a portion of the configuration data containing sensitive resources and a second readback of the same portion of the configuration data was performed to recalculate the CRC code and a comparison between the first and second checksum was performed for error detection. Another error detection technique involved a self readback process of the configuration memory and a comparison is performed on a duplicate copy. The correction technique involved reconfiguration of the affected data through scrubbing where the correct data is written back to the configuration memory. The first correction technique involved reconfiguration of the frame only with the upset bit rather than the entire design and the second technique involved a blind scrubbing, where a total reconfiguration of the configuration memory is performed periodically whether an upset is detected or not. However, frequent reconfiguration of the configuration memory from a duplicate copy irrespective of error increased the energy consumption of the scrubbing process because the SRAM cells was constantly accessed, thereby increasing the switching activity of the transistors in the cells which in turn increased the dynamic power consumption of the transistors.

Jonathan *et al.*, (2008) proposed an error detection technique for FPGAs called Duplication with Compare (DWC). This technique was effective at quickly and accurately detecting SEUs in the configuration memory. In the work 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 was stored in an external radiation memory, and comparator insertion for external system was deployed since it was placed at the final output of the two modules because it was less complex to be implemented and also saved cost. In an event of upset which was implemented in the work using a fault

injection simulator to create SEUs, the bitstream of the original module was compared with the golden copy by the comparator and discrepancy at the output of the two module, the comparator signals the system with an error flag indicated the presences of error. However, this mechanism was only able to detect errors and there was a delay from the time an upset occurs to the time when it was detected. The technique incurred more power consumption because it used the external interface to access the configuration memory.

Heiner *et al.*, (2009) proposed a technique that allowed Partial Reconfiguration (PR) to be used together with configuration memory scrubbing from a golden memory that explores external configuration interface. The scrubber utilizes portion a of the FPGA, performs the necessary operations to reconfigure a portion of the design while continuously scrubbing the entire FPGA. The system isolated the design into subsections by dividing it into static reconfigurable region which contains all the design critical bitstream that do not change during PR and dynamic reconfigurable region which contained non-critical component for re-configurability. The PR allowed portions of the design to be modified or upgraded after deployment. Anytime PR is performed, the golden configuration memory bitstream in the external memory no longer matches the logic within the FPGA therefore the external memory must also be updated to reflect the changes in the FPGA else the scrubber will detect many discrepancies between the expected bitstream and the bitstream on the FPGA. However, PR and scrubbing cannot have access to the configuration interface since PR will try to write to the configuration memory while the scrubber tries to read and write to it as well. In addition, the use of golden memory incur extra power consumption because it uses the external interface to access the configuration memory.

Lanuzza *et al.*, (2010) proposed a configuration memory scrubbing core used for internal detection and correction of radiation induced configuration single and multiple bit errors without requiring external scrubbing. The proposed approach makes FPGA devices able to self-repair SEU, whereby detection and correction were 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 work presented a compression scheme to reduce the memory resources, needed to store correction reference codes, thus making the technique easily scalable. The EDAC codes are pre-computed and stored in each sensitive frame capable of correcting up to quadruple bit upset, and error detection and correction process is performed by reading back sensitive frames. The sensitive frames are scanned one by one through internal readback operations and for each frame an EDAC code are internally computed and compared to the corresponding golden reference stored in the EDAC code memory. To perform comparison, the pre-computed EDAC code was decompressed and when an upset was detected, the corrupted data frame is corrected. The check operations are repeated cyclically by incrementing the frame address till the last frame was checked and the process starts all over. However, the EDAC code will be corrupted by single event upset since the circuit was embedded in the FPGA and those not have any form of redundancy.

Herrera-Alzu & Vallejo, (2011) proposed a self-reference scrubber for FPGA TMR system based. An external selectable microprocessor access port interfaces the bidirectional links with the three FPGAs and a three-way external voter was introduce for majority voting which was implemented in a radiation-hardened device. The scrubber periodically detects both single and multiple error and correction was performed by peer

frame comparison without needing to access a golden configuration memory. Three identical configuration memory frames are readback from the FPGA and the frame triplets are compared on a byte basis if frames are found identical, readback resumes and new frames are read back. If there was discrepancy in one of the frame, the dirty frame was scrubbed with a correct frame from one of the other two frames. The work did not used a golden memory for correction purposes it was mainly used for power-on configuration and single event functional interrupt when an single event upset causes a significant change in the functional operation of the FPGA. However, periodically scrubbing the configuration memory reduces power consumption but at the same time the use of golden memory in an event of a single event functional interrupt incurs extra power consumption since an external interface was used.

Akagic & Amano, (2012) presented a study of designing highly adaptable coprocessors using CRC on an FPGA bus. The work introduced an FPGA based architecture for accelerating CRC with 32 and 64 bits Generator Polynomials (GPs) in which a circuit was designed alongside a module for generating content of look-up tables. In order to reduce the number of logic resources used, the work explored possibility of using partial reconfiguration to minimize the area overhead when CRC standards are changed. The co-processor adapts to new CRC standards through the designed circuit. In the table generator module, in every clock cycle the counter generates input data, these data passes through the input reflection unit or the remainder generator, depending on the CRC standard and the input reflection unit reflects data bits by swapping them around its center and the data was shifted by the left depending on a data-path width and the length of the slice. The remainder generator unit performed long division operation. Number of cycles required to

generate one remainder depends on the offset of a byte in an input data. At the end, remainder is reflected or forwarded to output, depending on a CRC standard. In the CRC co-processor implementation, when creating reconfigurable region and the polynomial width is less than 64 bits, then the most significant bit are filled with zeros. However the CRC adaptable co-processor affects the area utilization despite the reduced resources used this is because integrating the table generation module with the CRC module requires additional area in the FPGA floor plan and as the table generation increases, it adds to the latency of the CRC calculation.

Nazar *et al.*, (2013) proposed a shifted scrubbing technique whenever an error occurred the scrubber starts scrubbing the associated partition where the critical bits in the configuration memory ere flipped after they were been detected to avoid functional failure of the device. A partitioned designed was use for the scrubbing technique and each partition was a Dual Modular Redundancy (DMR) as an error detection mechanism which indicated error the scrubber started scrubbing the associated partition after an upset was been detected. A fault injection platform was also implemented where faults were injected through the internal configuration access port mimicking single event upset. 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. The Drawback of the proposed technique was that the scrubber do not have knowledge of the optimum starting frame for each partition in an event of a single event upset thereby increasing error detection time as

the scrubber reads back the frames to get this information which will consequently impact negatively to the correction time.

Vera *et al.*, (2013) implemented an error detection and correction code core for femto architecture and its implementation does not required the use of a golden bitstream for scrubbing and also the method those not required any modification at the architectural level of the FPGA. The femto was a unit that controlled the scrubber and fetched data or command from an interface which supported a limited set of instructions focused on moving data from and to the memory under scrubbing. The work adopted a methodology of using Bose and ray-ChaudHuri (BCH) codes for error correction which involved addition of parity bits. Each frame is added a random sequence of 1's and 0's prior to encoding with the BCH coder to produced n check bits. The BCH encoder/decoder generated 57 check bits per frame and a CRC bit was also calculated and stored in a golden frame together with the BCH code for an uncorrupted frame. During the device operation where the frames were subjected to radiation, the scrubber reads the frame and correct any error using the BCH code, afterwards the CRC code was recalculated for the frame and if there was a mismatch with the reference CRC code in the frame, then this signifies an error in the frame and hence the frame has to be updated from a programmable read only memory which implied a penalty on latency and power consumption. However, implementing BCH encoder/decoder increase error correction latency since these added to the complexity of the scrubber because of the large data path it exhibited in generating the check bits.

Rao *et al.*, (2014) proposed a scrubbing scheme which reconstructs erroneous configuration memory frame based on the concept of erasure codes for MBU in the

configuration frames of FPGAs. The erasure codes recovered the original frame when some of the bits were flipped. The scrubbing technique was implemented as a soft module alongside with the user design and both were mapped into the FPGA. The work employed a low-cost Interleaved Two-Dimensional (I2D) parity technique to detect MBUs in the configuration memory frames of the FPGA. The interleaving distance was improved based on the actual MBU patterns because each horizontal parity bit is the exclusive-OR operation of the bits in the multiple row, all rows that were separated by a constant distance forms an interleaving group and all the bits within that group are covered by only one horizontal parity bit. A scrubber unit periodically checks the configuration frames to detect possible erroneous frames. Once an error was detected by assuming that the erroneous frame was erased, its contents were reconstructed using an erasure code by computing an exclusive-OR operation of all bits in the temporary block that were initialized with zero bits by the recovery unit, thus, the temporary block was written into the erroneous frame. The encoding of the erasure code was done once offline in advance during when the design was mapped into to the FPGA device. However, the erasure codes are also be subjected to single event upset since they can also embedded inside the FPGA without redundancy.

Wang *et al.*, (2014) proposed a circuit for detecting single particle flip failure for SRAM based FPGA. A dual redundancy alongside with CRC was used to reinforce the circuit. The designed circuit traversed the whole device address memory so as to detect failure early. Dual redundancy with cyclic redundancy check was used on the FPGA to mitigate SEU. A reference FPGA was used for comparison with a test FPGA. In the test system a data acquisition software communicated with the FPGA via the serial port to the First-In

First-Out (FIFO) module after the conveyed data was processed by the CRC module which encoded and decoded the data for error detection purposes by first receiving the data from the test board and then putting the data in the FIFO which is sent at the same time to the same address of the memory cell of the two SRAM FPGA, after which the CRC code in reference SRAM FPGA was checked. If the checksum was incorrect the data was discarded and the whole process repeated, but if the checksum was correct, the two SRAM FPGA data are read and outputs the comparison result after using the dual redundant module for error detection. The data processing starts from the first address of the test SRAM FPGA and reference SRAM memory through all the SRAM memory cells bit by bit. After the data was processed by the data processing module, the sensitive bits of SEU and its location can be determined. Then the information can be transported through the serial port to the data acquisition software for mitigation. However, the use of dual redundancy, FIFO module and CRC modules increased the energy consumption because these modules required the use of external interface to communicate.

Sanchez *et al.*, (2015) proposed a technique to mitigate SEU effect in FPGA. The work was based on an approach to build Partial TMR circuits for FPGAs using approximated logic circuits. The work presented a TMR circuit where the two redundant copies were built as an over approximated and under approximated copy of the original copy, whereby the approximated logic circuit performed a possibly different but closely related logic function, so that it can be used as a fault tolerant technique to mask error where it overlapped with the original circuit. Then, Partial TMR can be implemented by voting among approximated logic circuits instead of exact copies of the original circuit. The approximated logic circuits covers the most critical bits and reduced mitigation on bits that

were less critical to reduced resource utilization. However, since the partial TMR masks only the sensitive bit, some bits were prone to single event upset.

Jing *et al.*, (2015) proposed a Duplication With Recovery (DWR) technique to correct soft error in an FPGA configuration memory for bits meant for routing resources, which contributed to the majority of soft errors in FPGAs and are most vulnerable to single event upset because they occupy most area and seventy to ninety percent of the configuration bits were attributed to the routing resources. DWR leveraged redundant resources and enables fault recovery. The work efficiently duplicated a routing algorithm for DWR. Compared with traditional fault tolerant technique like TMR, DWR eliminated expensive copies when compared with TMR and required a the fault controlling circuitry; which controlled the fault signal on site in the routing Programmable Interconnect Points (PIPs), fault recovery structure; which recovered the correct signal with the duplication before the sink logic, and net duplication; which offered the opportunity for signal recovery. The duplicated nets were routed closely to its original net for convergence purposes. The work showed that the protection using limited resources can be sufficient for non-critical mission application. However, the technique only focused on mitigating SEUs in the routing resources indicting partial mitigation technique, while configuration bits in other FPGA resources may be corrupted.

Tonfat *et al.*, (2015) proposed a technique of correcting soft errors of a novel scrubbing technique using internal frame redundancy called FLR-scrubbing. This correction technique can be implemented in a coarse grain design. It was based on having each redundant TMR domain with the same configuration frame information to allow vote out the frames copies to restore themselves. The scrubber reads the first frame of each region

then executes a bit level majority voting on the three frames, the scrubbing technique enabled the self-correction of faulty frames using the information of the other two copies. In the methodology adopted in the work, the readback and scrubbing process were complementary because the readback process generated the correct frame that the scrubbing process needed for a possible write operation. Implementation results demonstrated minimum area and energy consumption overhead when compared to blind scrubbing technique, the scrubbing used the Internal Configuration Access Port (ICAP). However, the voting mechanism for correction was on a bit level basis which incurred longer correction time which led to increase energy consumption as the whole configuration memory frame has to be readback on bit level attempting to performing voting without any error detection mechanism.

Jorge *et al.*, (2015) proposed a scrubbing methodology that reduced energy consumption avoiding the use of an external memory to store the reference configuration bitstream. The work adopted a dynamic partial reconfiguration using internal frame redundancy known as FLR-scrubbing, which can correct multiple bit-flips. The Proposed technique relied on the generation of a TMR design which was possible to repair any frame of the TMR design using the information of the other two identical frames. The effectiveness of the scrubbing methodology to correct accumulation of bit upset in the configuration memory was analyzed by a fault injection experiment determined the maximum number of upset the mechanism can correct. The fault injector was implemented in the FPGA and the emulated upsets were a combination of random generated SEU locations. However, this mitigation technique required that the detection and correction mechanism were on a bit level for each

and complete frame which implied longer detection and correction time, which also impacted negatively on the energy consumption.

Aishwarya & Mahendran, (2016) proposed a technique to detect and correct MBU using erasure codes. The scrubbing scheme reconstructed the erroneous configuration frame based on the concept of erasure coding algorithm, here MBUs are detected by using the interleaving distance which was classified into horizontal and vertical parity. The erasure coding worked by creating a mathematical function to describe a set of numbers so that they can be checked for accuracy and recovered if one was lost; this was also known as polynomial interpolation. The erasure coding operated by making use of the decimal matrix code for the error detection and correction by creating a mathematical function to describe a set of numbers so they can be checked for accuracy and recovered if one was lost. Erasure coding technology transformed a message of k symbols into a longer message (code word) with n symbols such that the original message can be recovered from a subset of the n symbols. For detection parity bit were introduce to separate the bits with constant distance (interleaved parity, because both horizontal and vertical interleaving distance affects the detection coverage of the 12D parity technique) for area overhead minimization and to increase the detection coverage when the bits were distributed in several dimension against MBUs. MBUs were detected using the horizontal and vertical parity technique and then the erasure coding methodology assumed that the frame or subset of a logic block was erased when error was detected in the frame or logic block thereby the frame is recovered via the original frame. However, the use of self-repairable erasure code increases the time to repair of erroneous frame because the erasure coding first transformed a message of k

symbols into a longer message (code word) within symbols such that the original message can be recovered from a subset of the n symbols.

Wirthlin & Harding, (2016) presented 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 was built into the FPGA to compute a check word for each frame during configuration readback. The scrubber continuously reads configuration frames and computes a syndrome for the readback frame. The syndrome was compared with the internal ECC word of the frame determined whether the frame was error free, contains a single error or multi-bit error. This process continues through all frames of the configuration bitstream. After scanning the all block of frames, the process was repeated with the first frame in the configuration. If a single-bit error was found, the internal ECC circuitry computes the location of the error and corrects the upset. The internal scan unit then writes the corrected frame back into the configuration memory. Some multi-bit errors (three bits or more) within a frame may not be detected by the ECC. To detect this condition, a global CRC was provided for the entire set of frames. This CRC was recomputed during each full scan of the configuration memory and compared against an internal global CRC. If a multi-bit error occurred that was 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, therefore, the scrubbing system does not know the location of upset frame and the scrubber scrubs every frame. However, when multi-bit error was continuously detected this resulted to increase energy consumption because continuous device configuration will be required.

From the literatures reviewed, it is evident that Single Event Upsets (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. This research work is an improvement of Tonfat *et al* (2015), where FLR scrubbing was used to mitigate multiple errors within the configuration memory to ensure data integrity. The scrubbing methodology adopted from the literatures reviewed showed an inability of the techniques to scrub errors in FPGA configuration memory in an acceptable short period of time. Also the scrubbing process incurred higher energy overhead which has result to poor judgment in the trading-off of the parameters considered in scrubbing methodology. An improved FLR scrubbing algorithm with less detection time against SEUs was developed in this research work. The algorithm reduced the time required to mitigate SEUs which resulted to faster scrubbing time and reduced energy consumption. This implied that the improved algorithm ensured and guaranteed the reliability of the entire system, especially in critical application such as space technology where reliability is paramount.

CHAPTER THREE

MATERIALS AND METHODS

3.1 Introduction

This section describes the methods, materials and procedures employed for the successful completion of this research. The Frame Level Redundancy (FLR) Scrubbing Algorithm, the improved Frame Level Redundancy Scrubbing Algorithm based on Cyclic Redundancy Check (CRC) as an error detection technique were developed and simulated using MATLAB R2013a simulation environment.

The Methodology adopted in this research for the development of an improved FLR scrubbing algorithm based on CRC detection technique 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 R2013a 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. Development of an improved FLR scrubbing algorithm. 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. Performance comparison of the results obtained with that of the existing FLR scrubbing algorithm developed by Tonfat *et al.,* 2015 using error correction time and energy consumption as metrics for the purpose of validation.

3.2 Generation of Field Programmable Gate Array Configuration Memory.

There is no known MATLAB command capable of generating random binary array of numbers. 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. The MATLAB function code for implementation is in Appendix A.

The *rand* command generates random numbers between 0 and 1 ($0 < \phi > 1$), and the limiting factor η ensures that the random numbers does not exceed the value 1. In this work η was chosen to be 0.8 in other to obtain a realistic combination of 0 and 1 binary numbers, it should also be noted that the random numbers generated are rounded-up to the nearest integer of either 0 or 1 emulating the logic bits in the FPGA configuration memory module, if τ defines the size of the module with N as the number of rows representing the SRAM cells and D the number of columns representing the frames in the configuration

memory module. Therefore, the logical expression used to generate the configuration memory is written as shown in Equation (3.1).

$$M_n = \phi(\tau) > \eta \tag{3.1}$$

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.

Note: From the logical expression in 3.1, *M* represents Module and *n* ranges from 1 to 3, if *n* is 1, this implies that module 1 is being generated, similarly to module 2 and 3.

3.3 Configuration Memory Triplication

In other to provide redundancy, the configuration memory is triplicated into three modules to enable bit-level voting possible. Figure 3.1 shows a snippet MATLAB functions (Script) that was implemented, the complete program is shown in Appendix A.

```
MODUL=rand(N,D)>0.8;
MODUL1=MODUL;
MODUL2=MODUL1;
MODUL3=MODUL2;
```

Figure 3.1: Snippet of Configuration Memory Triplication Implementation.

From Figure 3.1, N and D represent the row and column size of the field programmable gate array configuration memory module respectively.

3.4 Methodology for a Fault Injection Campaign

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 user decides the choice of the model to which fault is to be injected) using Equation (3.2).

$$M_f = \begin{cases} M_1 \;\; If \;\; Module\, address\, is\, 1 \\ M_2 \;\; If \;\; Module\, address\, is\, 2 \\ M_3 \;\; If \;\; Module\, address\, is\, 3 \end{cases} \tag{3.2}$$

where:

M_f is the fault injected module

M_1, M_2 and M_3 represents Module 1, Module 2 and Module 3 respectively.

From equation 3.2, 1 is used to select M_1, 2 is used to select M_2 and 3 is used to select M_3 for fault injection, this is because in this work different methods of fault injection were

considered. Thus a technique which could be able to select a particular fault injection method becomes necessary.

The flow chart of fault injection is shown in Figure 3.2.

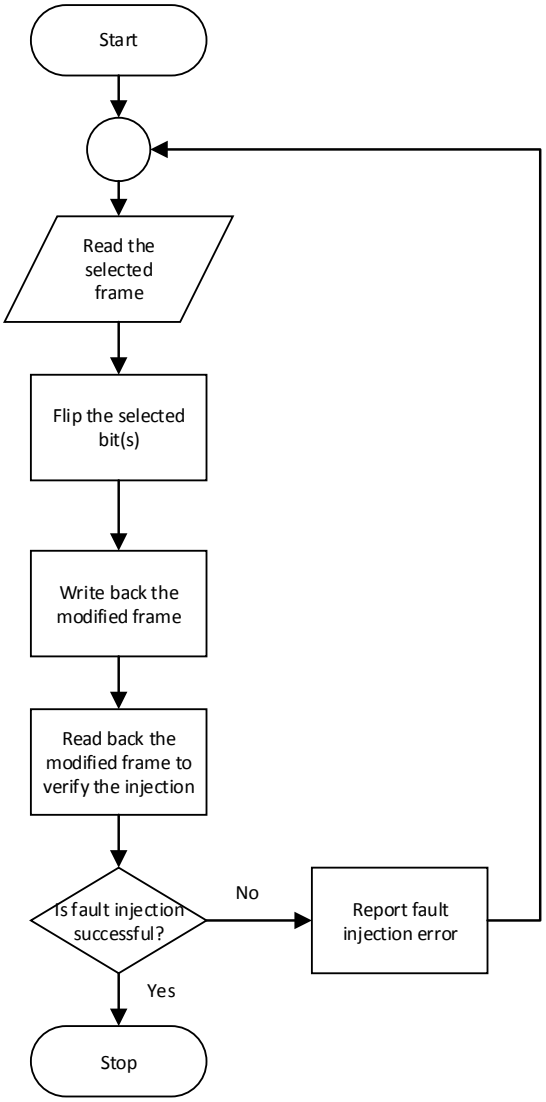


Figure 3.2: FPGA Configuration Memory Fault Injection.

Figure 3.2 shows the procedure for injecting fault(s) into the configuration memory. The information needed to flip bit(s) is the selected frame address and the bit(s) position or SRAM cell. It is important to mention that this technique can also emulate intra-frame and

inter-frame upset as well as single and multiple bit upset. Fault injection errors could arise due to the inexistence of the selected frame address.

During fault injection process from Equation (3.1), M is used to select the module on which fault is to be injected. For example, if n =1, then, the fault is injected in module one and so on.

$$[N, D] = \text{Module Size} \tag{3.3}$$

Equation (3.3) is used to determine the row and column size of the configuration memory module. To inject fault emulating single event upset in module one, the logical expression is used and expressed by Equation (3.4).

$$\text{mFault} = \sum_{i=1}^{N} \begin{cases} \text{Module} = \text{Module in Error} & \text{if} \quad m = 1 \\ \text{Module} = \text{Module1} & \text{otherwise} \end{cases} \tag{3.4}$$

mFault represents the module with fault, and m = 1 this implies that only module 1 is with error. Module 1 otherwise implies that if $m \neq 1$ meaning module 1 is not selected to be injected with fault hence, module 1 should be unchanged and its original configuration be kept. The summation symbol implies that the configuration memory module with its elements from row 1 to N should be picked one after the other to be decided whether to be flipped or not. This logical principle applies to when fault is injected in only two modules or to all the three modules, whereby the errors are randomly generated based on the specified region selected in the FPGA configuration memory. The same procedure is used to inject fault in module 2 and module 3 only.

Similarly, to inject fault in two modules, the logical expression is used and expressed by Equation (3.5).

$$\text{mFault} = \sum_{i=1}^{N} \begin{cases} \text{Module} = \text{Module in Error} & \text{if} \quad m = 1 \\ \text{Module} = \text{Module in Error} & \text{if} \quad m = 2 \\ \text{Module} = \text{Module1, Module 2} & \text{otherwise} \end{cases} \tag{3.5}$$

Equation (3.5) is used to inject fault in module 1 and module 2. The same procedure is used to inject fault in module 1 and 3 or module 2 and 3.

Similar to Equation (3.4), to inject fault in three modules, the logical expression is used and expressed by Equation (3.6).

$$\text{mFault} = \sum_{i=1}^{N} \begin{cases} \text{Module} = \text{Module in Error} & \text{if} \quad m = 1 \\ \text{Module} = \text{Module in Error} & \text{if} \quad m = 2 \\ \text{Module} = \text{Module in Error} & \text{if} \quad m = 3 \\ \text{Module} = \text{Module1, Module 2, Module 3,} & \text{otherwise} \end{cases} \tag{3.6}$$

Note that in each of this cases, the errors are generated randomly with different values. The MATLAB function code to implement fault injection is in Appendix B.

3.5 Majority Voting Implementation

This technique relies on the generation of a 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 research, the voting is realized using the logical expression shown in Equation (3.7).

$$M_v = \sum_{i=1}^{\text{length}(m)} \left\{ \begin{array}{c} M_1(:,:), M_2(:,:), M_3(:,:) \quad \text{if} \quad M_1(:,:) = M_2(:,:) \;\&\; M_1(:,:) = M_3(:,:) \\ M_1(:,:), M_2(:,:) \quad \text{if} \quad M_1(:,:) = M_2(:,:) \;\&\; M_3(:,:) \neq M_1(:,:) \text{ then } M_3(:,:) \Leftrightarrow M_1(:,:) \end{array} \right\} \tag{3.7}$$

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.

The summation symbol implies that the module elements which are in rows and columns are converted into string of binary array of length m. In the first case, considering the elements of module 1,2 and 3, if the elements of module 1 are the same with that of module 2, and also the elements of module 1 are the same with that of module 3, therefore, bit-level voting will not be activated since all the three modules have the same bits. However, if one out of the three modules have different logic bits in the same frame address and at exactly the same cell position in the same scrub cycle. For example in the second case in Equation (3.7), module 3 is with error hence is voted out using module 1.

3.5.1 Frame Level Redundancy Scrubbing Process

This mechanism was used to enables the correction of accumulated faults or upset in the FPGA configuration memory and these upsets includes both SEUs and MBUs. The redundant modules (module 2 and 3) were generated internally to the FPGA and it was ensured that the three modules consist of exactly the same configuration memory frames and logic bits combination. Two frames having the same frames address in the TMR were used to correct any upset during the read-back process of the FPGA. The first frame of each TMR domain were read during the scrubbing circle, after which the scrubber executes a bit level majority voting when an upset is detected in any of the three frames therefore creating a fault-free (voted) frame, and then the scrubber moves to the next frame position of each TMR domain with the same frame address.

This research assumes both the best and worst case scenarios of scrubbing. In the best case scenario, none of the frames having the same frame address needs to be scrubbed, therefore the scrubber moves to the next frame address, when a scrubbing process occurs. This implies that only one bit is in error (upset) at exactly the same cell position having the same frame address in the TMR domain, which is the correct execution of the scrubbing when at most one bit is in error. In a worst-case scenario, three frames with the same frame address are corrupted at exactly the same cell position in a scrub cycle due to the accumulation of bit upsets, but this is very unlikely to occur in a scrub cycle considering the enormous large logic resources, configuration memory bits in the FPGA and the scrub rate and also considering the stochastic nature of single event upset in radiation environment such as space.

Therefore, the FLR scrubbing process is based on an assumption that the probability that an upset will occur in an FPGA configuration memory at exactly the same cell position in frames with the same frame address in a scrub cycle is very negligible. In addition, the FLR scrubbing process assumes that the time it will take to read a frame of data in the configuration memory is the same as the time it will take to detect error in that frame, which is not true, therefore excluding error detection in the process or assuming error has already been detected. However, even if such a scenario occurs a full device re-configuration is activated. Figure 3.3 shows bit-level majority voting for correcting frames with identical frame address in an FLR scrubbing as implemented in this research.

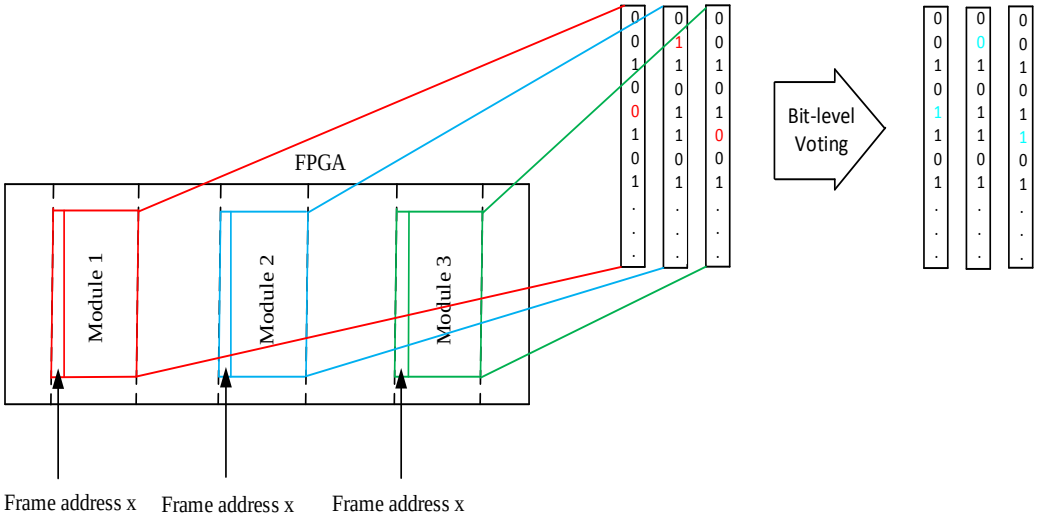


Figure 3.3: Bit-level Majority Voting in an FLR Scrubbing Technique.

The Frame Level Redundancy scrubbing process in FPGA is controlled by a voting algorithm performing a two out of three bit-level voting is shown in Figure 3.4.

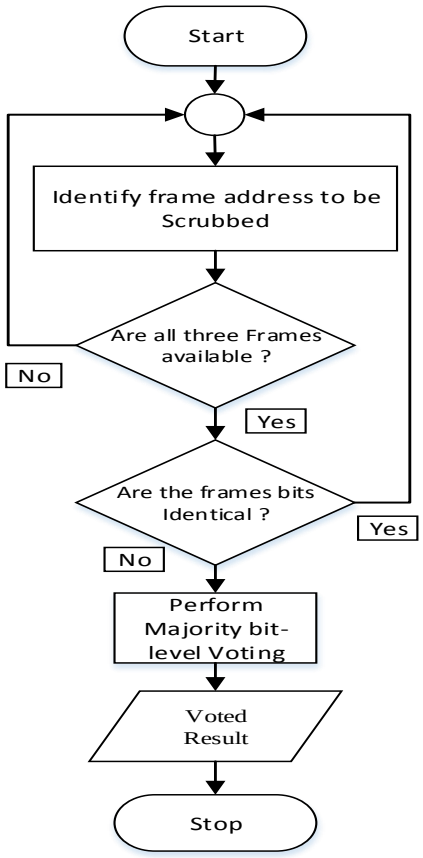


Figure 3.4 Flow Chart of a Voting Process.

3.6 Majority Voter

Majority voter plays an important role in ensuring the correct operation of the scrubbing process. This research made an assumption based on an idle or fault tolerant majority voter is adopted to ensure that the voting process is without error. If the inputs to the majority voter from the same frame address from module1 represents input A, module 2 represents input B, module 3 represents input C and the fault tolerant majority voter output is given by Y, then the truth table in a TMR domain in a bit-by-bit majority voter logic circuit is given in Table 3.1.

Table 3.1 Majority Voter Truth Table

A	B	C	Y	Error Signal
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	1
1	0	0	0	1
1	0	1	1	1
1	1	0	1	1
1	1	1	1	0

Logic equation for majority voting circuit in a TMR domain are written as follows:
The Boolean logic equation for a possible combination when the bit with upset is a logic 1 is giving by Equation (3.7).

$$Y = \overline{A}\overline{B}C + \overline{A}B\overline{C} + A\overline{B}\overline{C} \tag{3.7}$$

The Boolean logic equation for a possible combination when the bit with upset is a logic 0 is given by Equation (3.8).

$$Y = \overline{A}BC + A\overline{B}C + AB\overline{C} \quad (3.8)$$

The Boolean logic equation representing a possible combination for an upset indicating an error signal is given by Equation (3.9).

$$Y = \overline{A}\overline{B}C + \overline{A}B\overline{C} + \overline{A}BC + A\overline{B}\overline{C} + A\overline{B}C + AB\overline{C} \quad (3.9)$$

The Boolean logic equation representing a possible combination for no upset therefore no error indicating a fault free frame is given by Equation (3.10).

$$Y = \overline{A}\overline{B}\overline{C} + ABC \quad (3.10)$$

Figure 3.5 shows the snippet of the bit-level voting the complete implemented MATLAB code is shown in appendix D.

```
[n,m]=size(A);
Original=A;

for i=1:n
    for j=1:m
        tic;if A(i,j)==B(i,j) & A(i,j)~=C(i,j);T_Frame=toc;
            tic;C(i,j)=A(i,j);T_Correct=toc;
            Voted_Module=C;
            end
        tic
        if A(i,j)~=C(i,j)&B(i,j)~= C(i,j)
        T_Frame=toc;
            tic
            A(i,j)=C(i,j);
            B(i,j)=C(i,j);
            T_Correct=toc;
            Voted_Module1=A;
            Voted_Module2=B;
```

Figure 3.5: Snippet of Bit-Level Voting.

3.7 Cyclic Redundancy Check

In this research the CRC was computed using the mathematical Equation (2.5) of Chapter 2. Figure 3.6 shows the snippet for generating CRC. The complete developed MATLAB function code implementation is shown in Appendix C.

```
function OUT=CRC(A,B,C,D)
[r,c]=size(A);
for i=1:r
    for j=1:c
        if A(1:end,1)==B(1:end,1)
            if A(1:end,1)==C(1:end,1)
                if B(1:end,1)==C(1:end,1)
                    if D(1:end,1)==A(1:end,1)
                        if D(1:end,1)==B(1:end,1)
                            if D(1:end,1)==C(1:end,1)
                                ww=A
                                XX=B;
```

Figure 3.6: Snippet of Cyclic Redundancy Check Implementation.

3.7.1 Cyclic Redundancy Check Frame Execution

During the readback process of the FPGA configuration memory frames for a possible detection of SEU for the purpose of scrubbing, the configuration memory frame bits of the test module in the TMR domain was read and a CRC code was calculated for the frames. During the next readback a new CRC value is generated for each data-frame that is read back and compared to the expected CRC result, if the CRC code matches it signifies an error free frame else, the frame is scrubbed to correct the upset as discussed in section 2.2.8 of chapter 2. Although it is very unlikely to have frames with the same frame address and SRAM cell position in the TMR domain containing SEU within one readback cycle and it should be noted that CRC check bits are small compared to the frame size therefore the CRC check code forms part of the frame header.

3.8 Improved Frame Level Redundancy Scrubbing Algorithm Implementation.

CRC exhibits fast detection time but lacks correction ability as compared to FLR. Therefore, the improved FLR algorithm hybridized CRC and FLR for error detection and correction, noting the fact that the FLR scrubbing algorithm attempts to perform bit-level voting irrespective of an error occurring or not.

In the improved FLR scrubbing algorithm, CRC technique is introduced as an error detection technique against SEU in the FPGA configuration memory. CRC is performed on the FPGA configuration memory test module for each frame in the first round of readback process.

In the absence of error, a switching mechanism is activated to alternate between CRC and FLR for every round of readback process. Fault-free redundant modules are needed for scrubbing in an event of SEU on the test module. Therefore, to ensure a fault-free redundant module, the FLR also corrects errors that might have occur in the redundant modules.

Figure 3.7 shows the flow chart implementation of the Improved FLR Scrubbing technique.

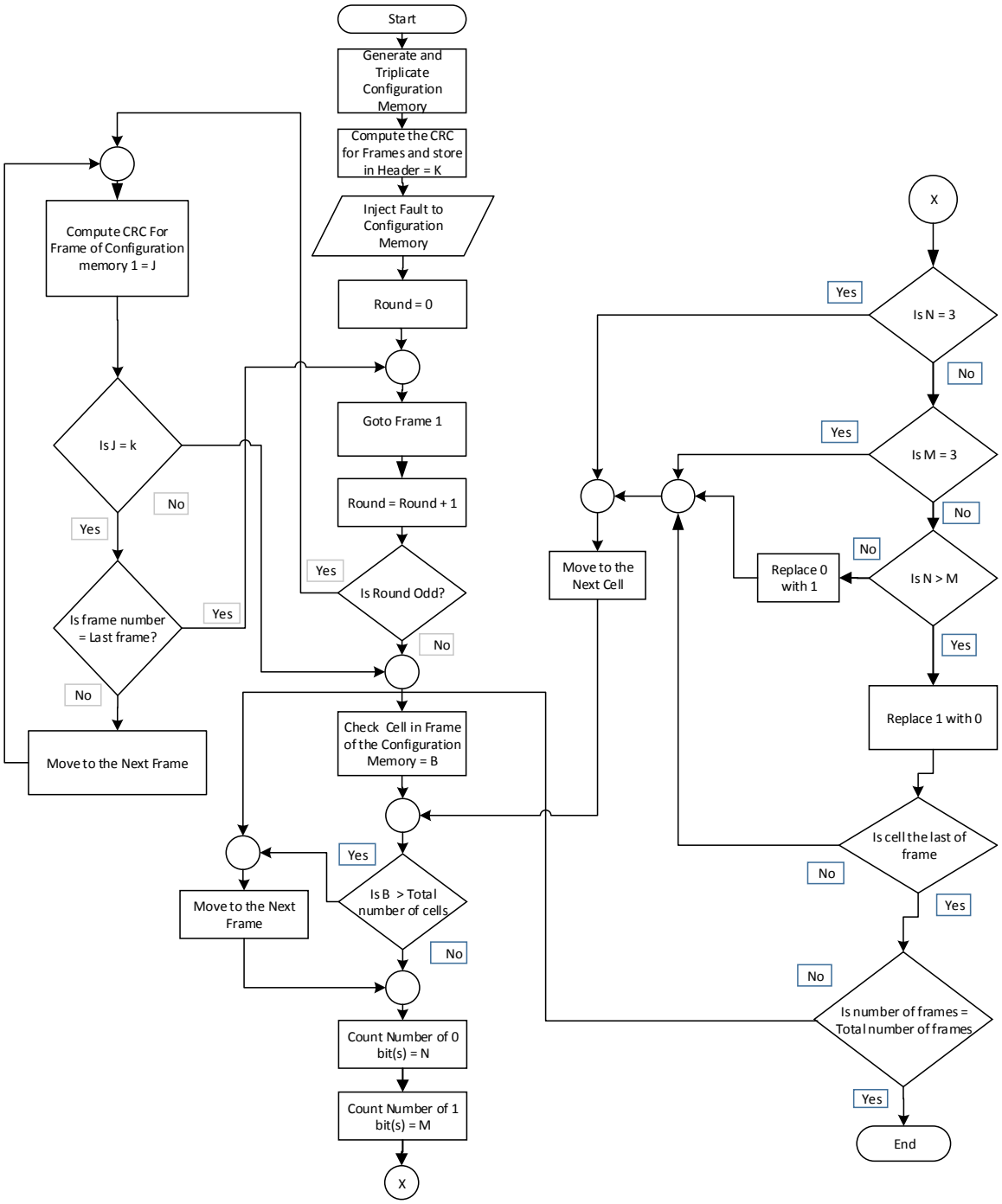


Figure 3.7: Improved Frame Level Redundancy Scrubbing Flow Chart.

3.9 Energy Consumption

The power consumption as discussed in section 2.2.10 of chapter 2 still depends on the scrubbing methodology and scrub rate or readback process. 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 (Tonfat *et al*., 2015) and expressed by Equation (3.14).

$$E_{Scrub_frame} = P_{Scrubbing} \times t_{Scrub\ rate} \tag{3.14}$$

where:

E_{Scrub_frame} is the energy required to scrub the configuration memory frames.

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

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

The power to scrub the configuration memory frame is given by Equation (3.11) according to Tonfat *et al* (2015).

$$P_{Scrubbing} = V_{core} \times I_{core} \tag{3.15}$$

where:

V_{core} is the internal supply voltage.

I_{core} is the maximum current.

CHAPTER FOUR

RESULTS AND DISCUSSION

4.1 Introduction

This chapter presents simulation results and discussion. The performance of the Frame Level Redundancy (FLR) scrubbing algorithm and that of improved FLR scrubbing algorithm employing cyclic redundancy check as an error detection in static random access memory based field programmable gate array are discussed and relevant results reported. A number of charts describing fault injection on different modules emulating single event upset in field programmable gate array during simulation are generated and discussed. Error correction time which is the time taken to scrub against Single Event Upset (SEU) and energy consumed during the scrubbing process were used as performance metrics to evaluate the FLR scrubbing algorithm performance.

4.2 Fault Injection Result

The FPGA configuration memory was generated in MATLAB 2013a simulation environment using the logical expression developed in section 3.2 of Chapter three. Figure 4.1, shows the generated configuration memory of the FPGA with different logic resources being subjected by SEU in only one module by the improved FLR. Table 4.1 shows the parameters used for fault injection in only one module and these parameters indicates the location size where the faults are injected in the module. The parameters from Table 4.1 used to generate Figure 4.1 was obtained by dividing the configuration memory module used by Tonfat *et al.,* 2015 by three (3) for simple representation and explanation. The parameters used by Tonfat *et al.,* 2015 for fault injection campaign experiment are 39

frames in a module, 126 SRAM cells per frame, 36,18 fault injection matrix size, 16,33 starting and ending frame address, 1,36 starting and ending SRAM cell address.

It should be noted that fault(s) can be injected in any part of the configuration memory module(s) depicting the randomness nature of SEU in real life scenario. The three modules forming the configuration memory are superimposed as shown in Figure 4.1 in two-dimensional view. The red spot indicates the SRAM cell(s) in the frame with error as a result of SEU. Thus, the fault injection matrix implemented in this research work defines the size and location of the fault injection campaign.

Table 4.1 Fault Injection Parameters for 1 Module.

S/N	Simulation Parameters	Value
1	Number of frames per module	13
2	Number of cells per frame	42
3	Number of module with fault	1
4	Starting frame address with fault	6
5	Ending frame address with fault	11
6	Starting cell address with fault	1
7	Ending cell address with fault	12

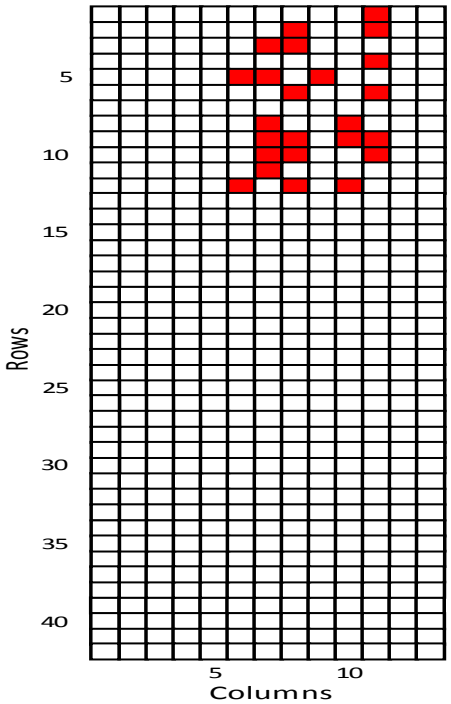


Figure 4.1 Single Event Upset in 1 FPGA Module.

Similarly, Figure 4.2 shows the SRAM cells in the configuration memory module with error after performing a fault injection campaign. However, this scenario is when faults are injected in any combination of two modules (i.e. 1 &2, 1&3 and 2&3). The parameters used to generate Figure 4.2 is given in Table 4.2

Table 4.2 Fault Injection Parameters for 2 Modules.

S/N	Simulation Parameters	Value
1	Number of frames per module	13
2	Number of cells per frame	42
3	Number of module with fault	2
4	Starting frame address with fault	3
5	Ending frame address with fault	11
6	Starting cell address with fault	11
7	Ending cell address with fault	22

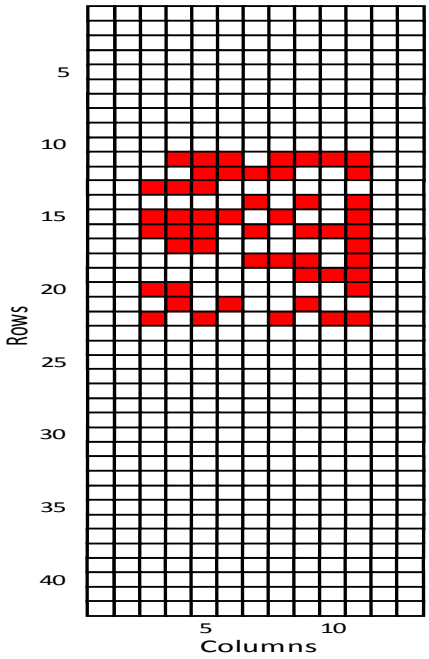


Figure 4.2 Single Event Upset in 3 FPGA Modules.

Similarly, Figure 4.3 shows when error is injected in all the three FPGA configuration memory modules. The parameters used to generate Figure 4.3 is shown in Table 4.3.

Table 4.3 Fault Injection Parameters for 3 Modules.

S/N	Simulation Parameters	Value
1	Number of frames per module	13
2	Number of cells per frame	42
3	Number of module with fault	3
4	Starting frame address with fault	1
5	Ending frame address with fault	10
6	Starting cell address with fault	29
7	Ending cell address with fault	40

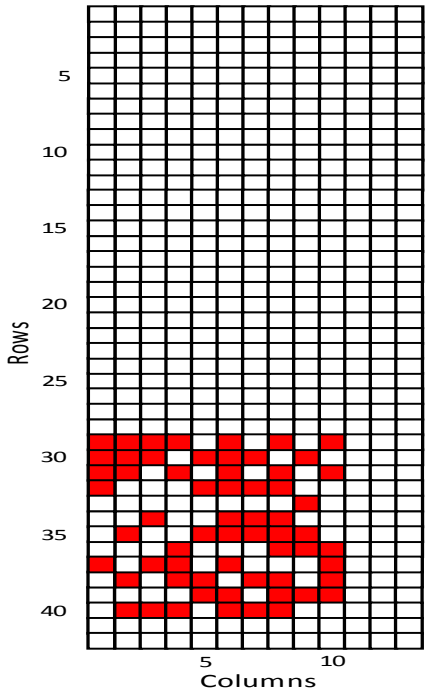


Figure 4.3 Single Event Upset in 3 FPGA Modules.

Figures 4.1, 4.2 and 4.3 were plotted from the values in Tables 4.1, 4.2 and 4.3 respectively, and Table 4.1, 4.2 and 4.3 shows the different method of fault injection was obtained from equation 3.4, 3.5 and 3.6 respectively. It should be noted that, in all the techniques the red spot indicates the SRAM cell(s) in the configuration memory module(s) with error.

It can be concluded from these Figures that, the number of SEU in the FPGA configuration memory increases as the number of module injected with faults increases. The total number of SEU in the FPGA configuration memory is more when two modules were affected compared to when it is one module and it increases when the three modules are affected.

4.3 Results of One Module Fault Injection

Table 4.4 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 4.4 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 results were obtained using the methodology as discussed in chapter three. 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.

Table 4.4: Time and Energy Consumption for 1 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	7.121	13.689	136.870

4.3.1 Error Detection versus Module Size

Figure 4.4 shows the graph of error detection time against number of frame in a module, the graph was plotted using MATLAB 2013a simulation environment from the simulation parameters and results obtained in Table 4.4 and was implemented with the MATLAB script in Appendix C.

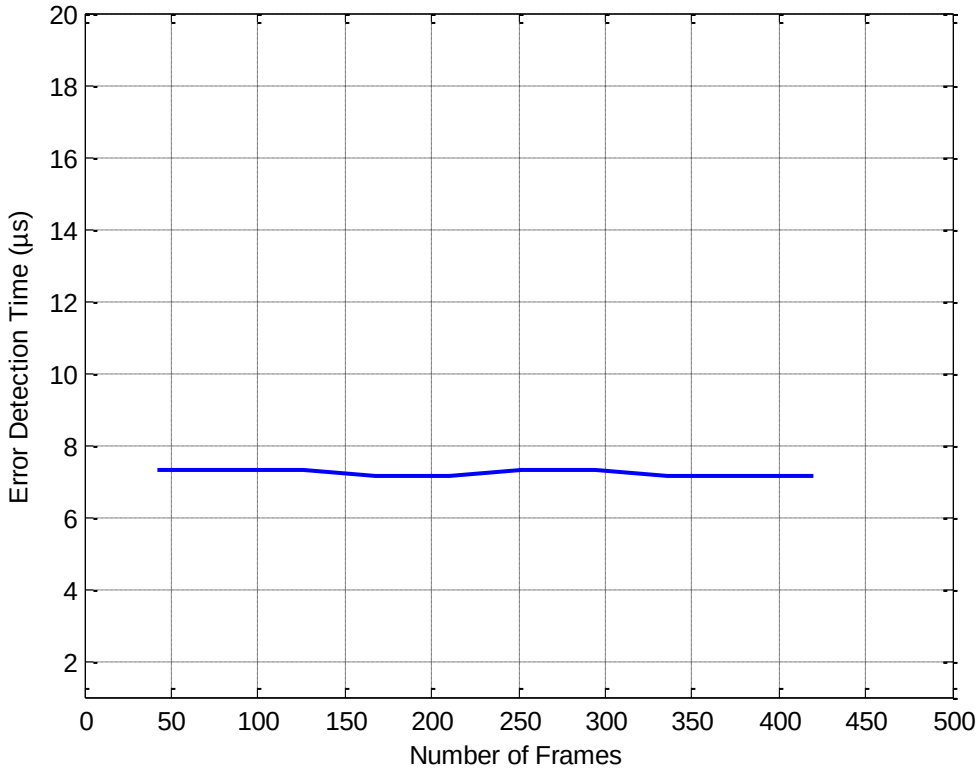


Figure 4.4: Error Detection Time versus Number of Frames for 1 Module Fault Injection.

From Figure 4.4, 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 was 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 do not exceed 7.333 micro seconds. The slight variations in the detection time is because of the inability to inject constant increment of errors in a module as the number of frames and

cells increases in a module. The same trend was observed when the fault injection was in two modules only and in all the three modules.

4.3.2 Error Correction versus Module Size

Figure 4.5 shows the graph of error correction time against number of frame in a module, the graph was plotted using MATLAB 2013a simulation environment from the simulated parameters and results obtained in Table 4.4 and was implemented with the MATLAB script in Appendix D.

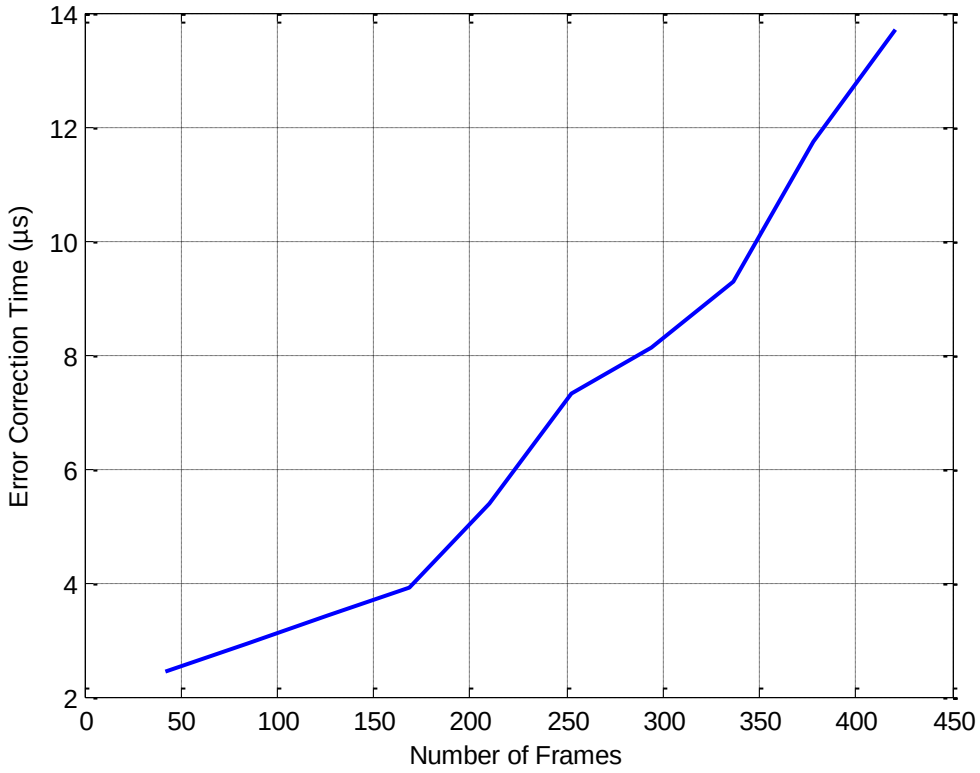


Figure 4.5: Error Correction time versus Number of Frames for 1 Module Fault Injection.

From Figure 4.5, 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 indicated 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. It was observed that 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. In addition, Figure 4.5 is not a linear graph because the number of injected in a module those increase with a constant value as the number of frames and cells increases in a module because the errors were randomly injected emulating the stochastic nature of SEU. This phenomenon is also applicable to fault injection in only two modules and in three modules.

4.3.3 Energy Consumption versus Number of Frames

Figure 4.6 shows the bar chart for the energy consumed versus number of frames in a module. The graph was plotted using MATLAB 2013a simulation environment from the simulated parameters obtained in Table 4.4.

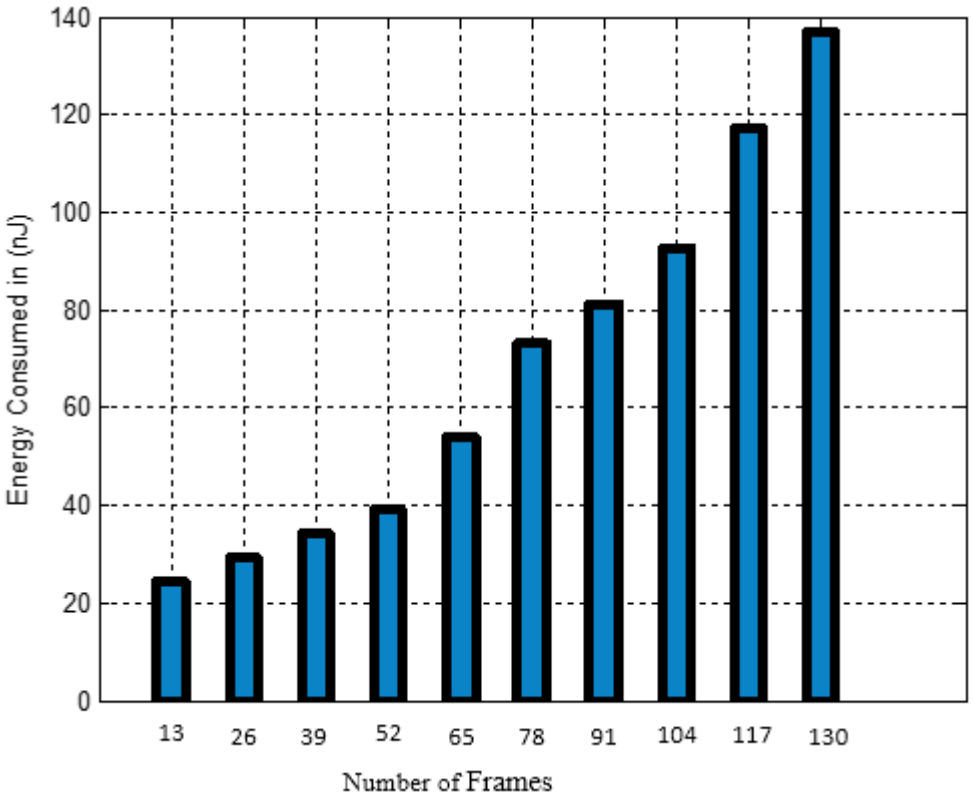


Figure 4.6: Energy Consumption versus number of frames for 1 Module Fault Injection.

Figure 4.6 shows the plot of energy consumed against number of frames in one module with fault injection. It can be seen from Figure 4.5 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 indicated that energy is a function of time at constant power as shown in Equation (3.14). Therefore, the values obtained from Table 4.4 for energy to scrub frames was the product of the error correction time and the operating power of the FPGA, where the power is the product of the FPGA operating voltage and current as given in Table 4.4. 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.

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). Similarly, Tables 4.5 and 4.6 showed the results when fault is injected in 2 and 3 modules respectively. 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, although this scenario is very unlikely to occur considering the enormous configuration logic bit in the FPGA and the stochastic nature of SEU.

Table 4.5 shows the results of error detection time, error correction time and the energy consumed during the scrubbing process when faults were injected in only two modules. The results were obtained using the methodology as discussed in chapter three.

Table 4.5: Time and Energy Consumption for 2 Module Fault Injection.

Virtex-5 FPGA							
Number of module with fault				2			
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	7.421	2.933	29.330
84	26	24,12	1,24	11,22	7.333	3.911	39.107
126	39	36,18	1,36	16,33	7.114	4.021	40.213
168	52	48,24	1,48	21,44	7.333	5.333	53.326
210	65	60,30	1,60	26,55	7.333	7.821	78.214
252	78	72,36	1,72	31,66	7.333	9.288	92.879
294	91	84,42	1,84	36,77	7.421	14.665	146.650
336	104	96,48	1,96	41,88	7.333	17.598	175.980
378	117	108,54	1,108	46,99	7.114	22.915	229.148
420	130	120,60	1,120	51,110	7.121	26.274	262.740

Figure 4.7 shows the graph of error detection time against number of frame in a module when fault was injected in two module, and Figure 4.8 shows the graph of error correction time against number of frames when fault was injected in two modules. The two plots were obtained from Table 4.5 and plotted in MATLAB 2013a simulation environment using the simulation parameters and results obtained from Table 4.5.

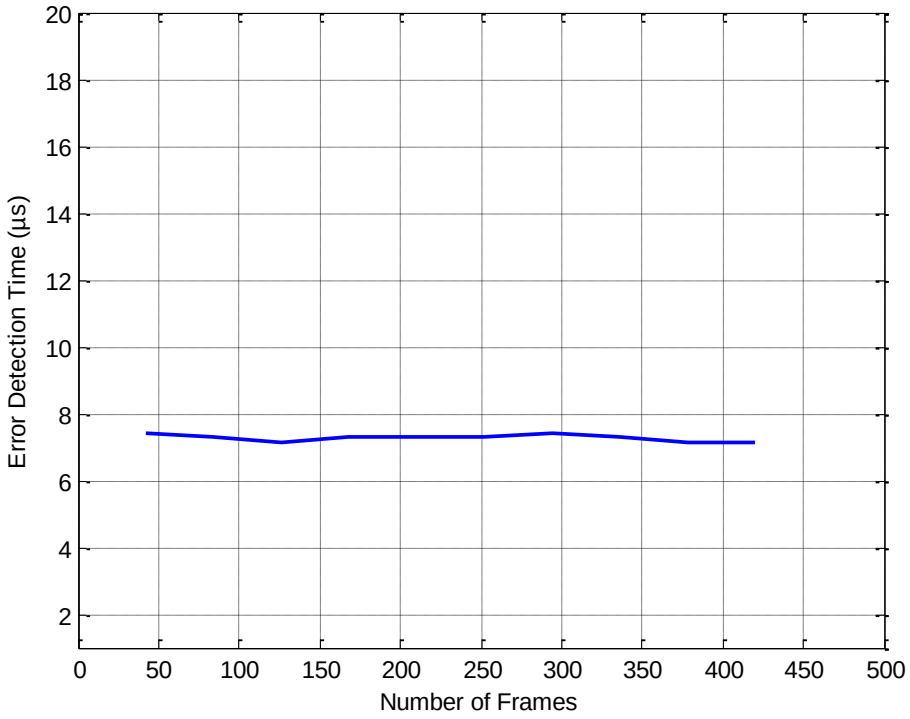


Figure 4.7: Error Detection Time versus Number of Frames for 2 Module Fault Injection.

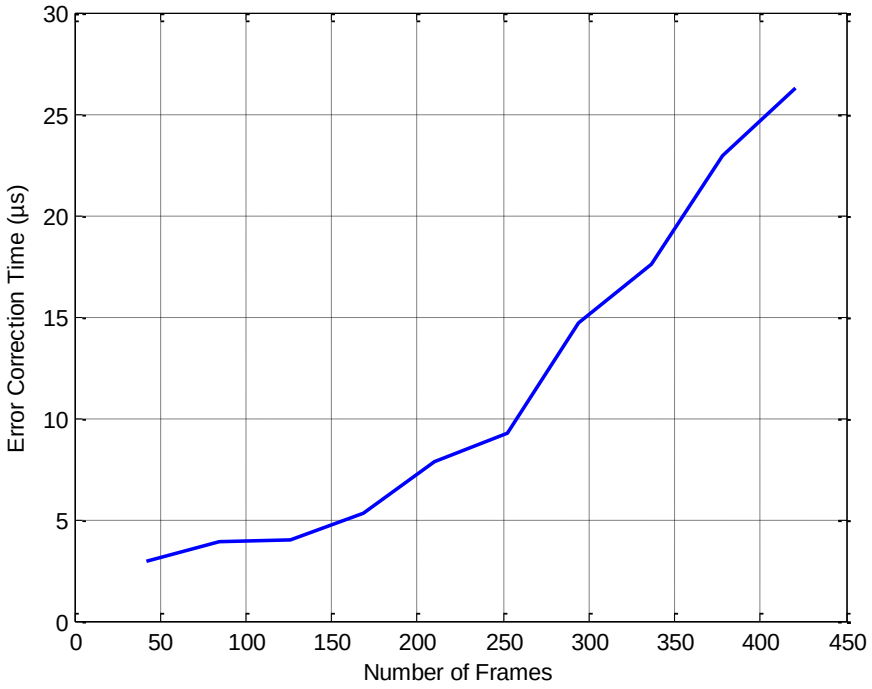


Figure 4.8: Error Correction Time versus Number of Frames for 2 Module Fault Injection.

From Figure 4.7, it was also observed that irrespective of the number of frames in a module the error detection time was within the range of 7.114 and 7.421 micro seconds, which is similar to Figure 4.4 when only one module is injected with fault although in this case the error detection time do not exceed 7.421 micro seconds. In Figure 4.7 it can be seen that the time taken to correct error increases as the number of frames in a module increases. However, the correction time is higher when fault is injected in two module than in one module for the same module size because the number of injected fault in two module is higher than that of one module.

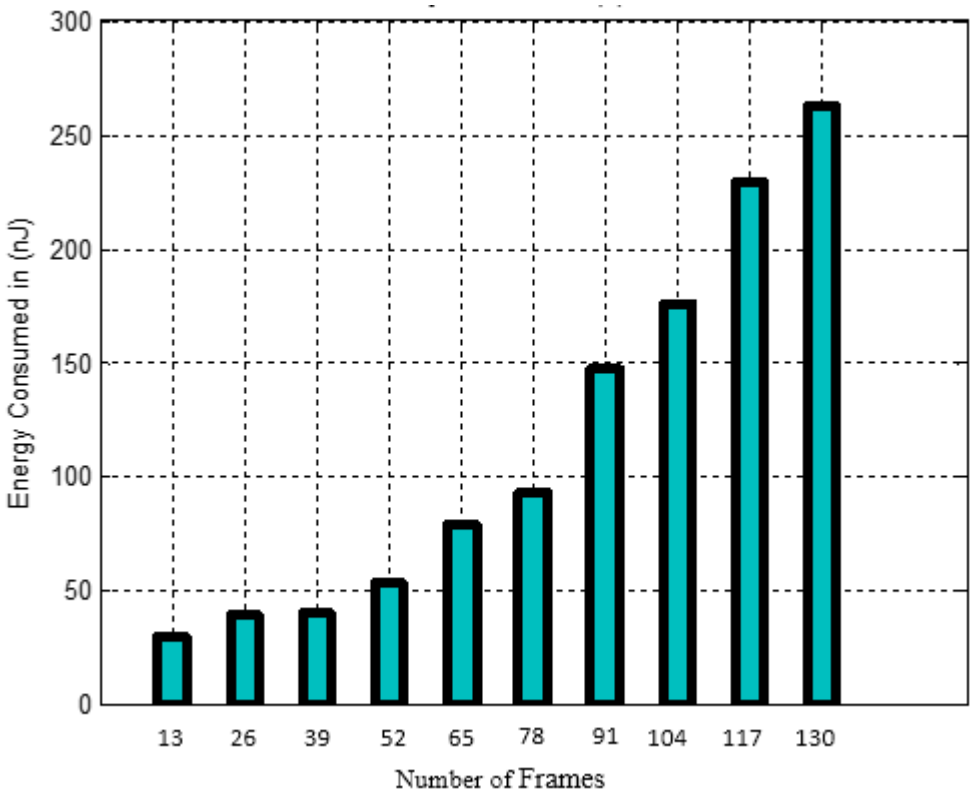


Figure 4.9: Energy Consumption versus Number of frames for 2 Module Fault Injection.

Figure 4.9 is a plot of energy consumed versus number of frames. It is similar to Figure 4.6. However, the energy to scrub a module with corresponding number of frames in Figure 4.9 was higher in each case when compared to Figure 4.6 because the time to scrub

two modules with error was higher compare to one module with both module having the same number of injected fault.

Table 4.6 shows the results of error detection time, error correction time and the energy consumed during the scrubbing process when fault was injected in all the three modules. The results were obtained using the methodology as discussed in chapter three.

Table 4.6: Time and Energy Consumption for 3 Module Fault Injection.

Virtex-5 FPGA							
Number of module with fault				3			
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	7.421	2.933	29.330
84	26	24,12	1,24	11,22	7.333	4.400	43.995
126	39	36,18	1,36	16,33	7.333	4.889	48.884
168	52	48,24	1,48	21,44	7.333	5.866	58.660
210	65	60,30	1,60	26,55	7.114	9.288	92.879
252	78	72,36	1,72	31,66	7.165	12.432	124.318
294	91	84,42	1,84	36,77	7.333	15.154	151.540
336	104	96,48	1,96	41,88	7.333	18.087	180.870
378	117	108,54	1,108	46,99	7.332	23.464	234.640
420	130	120,60	1,120	51,110	7.332	29.146	291.146

Similarly, Figure 4.10 shows the graph of error detection time against number of frame in a module when fault was injected in the three modules, and Figure 4.11 shows the graph of error correction time against number of frames when fault was injected in three module. Similarly, the two plots are obtained from Table 4.5 and plotted in MATLAB 2013a simulation environment from the simulation parameters and results obtained from Table 4.5.

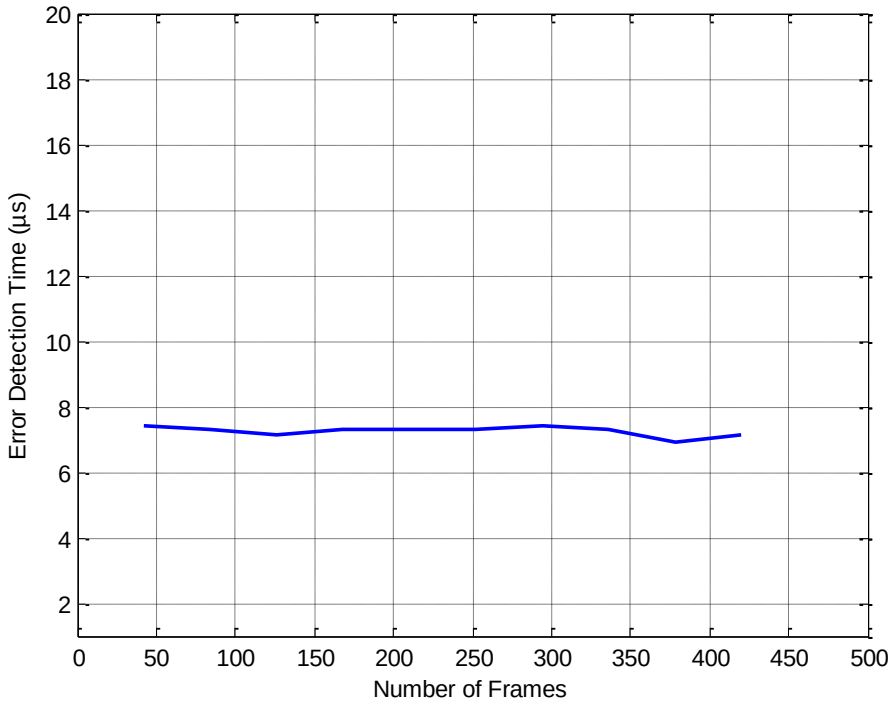


Figure 4.10: Error Detection time versus Number of Frames for 3 Module Fault Injection.

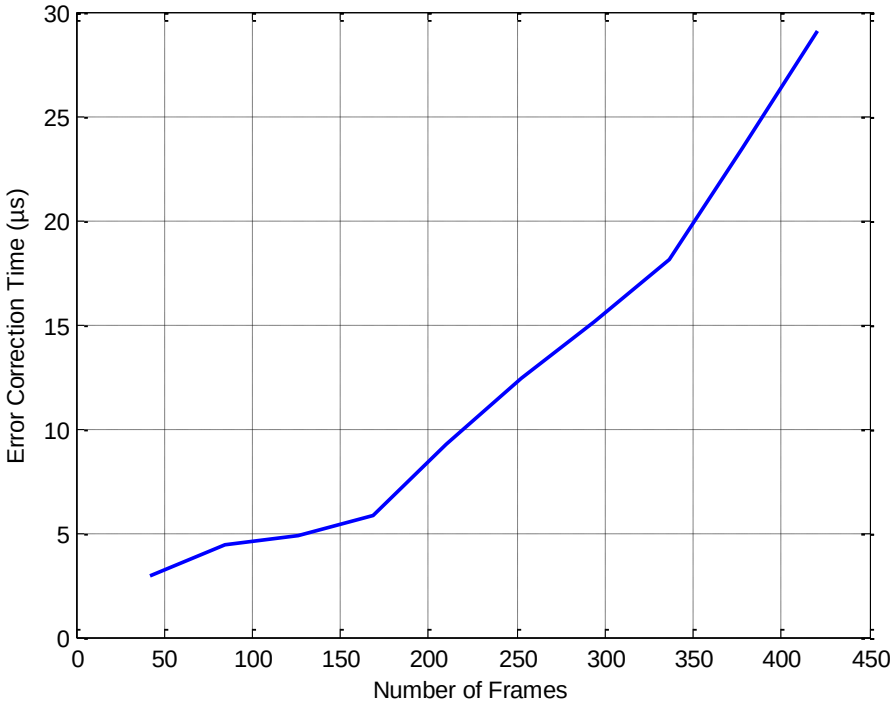


Figure 4.11: Error Correction time versus Number of Frames for 3 Module Fault Injection.

It was also observed in Figure 4.10 that the time taken to detect error when all the three modules were injected with fault is within the range of 7.114 and 7.421 micro seconds irrespective of the number of frames in a module and the density of the injected fault. This was similar to when fault was injected in only one and in two modules. Similar to Figures 4.5 and 4.8, Figure 4.11 shows that as the number of frames increases the time to scrub the frame also increases because the number of injected fault increases.

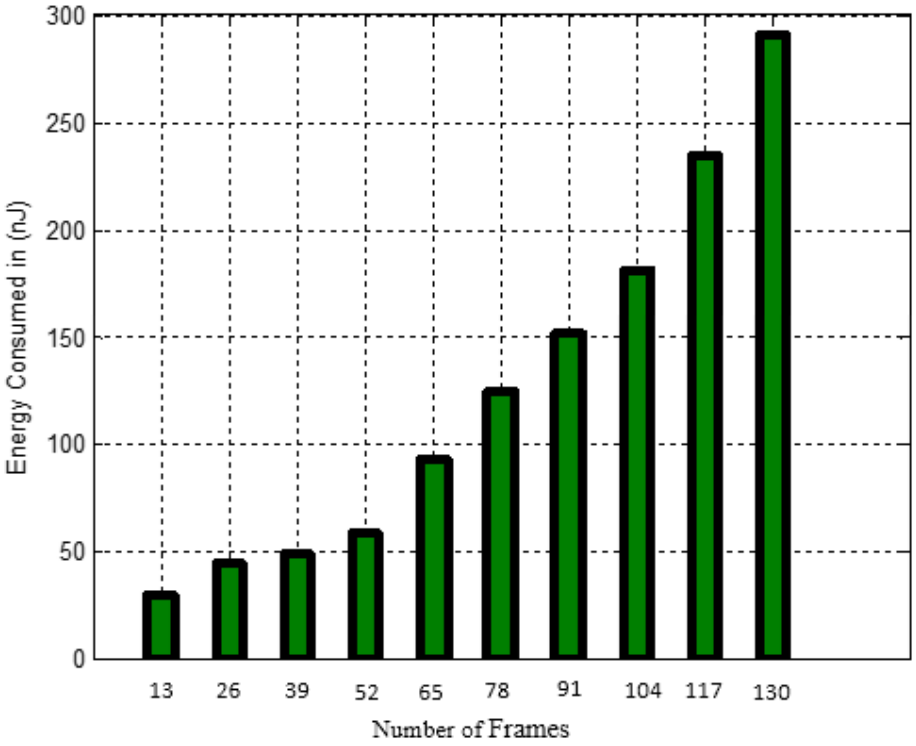


Figure 4.12: Energy Consumption versus Number of Frames for 3 Module Fault Injection.

In Figure 4.12, it can be seen that as the number of frames increases in a module the energy to scrub the frames increases because the number of injected fault also increases, this is similar to Figure 4.6 and 4.9. It was observed that the energy consumed to scrub when error were injected in three modules was higher compare to when it was in one and two modules.

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 occurs as when fault was 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 was because the total number of injected fault in the FPGA configuration memory increases.

4.4 Comparison of Improved FLR and Existing FLR Scrubbing Algorithm

Table 4.7 shows the comparison for the results obtained between FLR and improved FLR scrubbing algorithm using error correction time and energy consumption as performance metrics. The error correction time and energy consumption during the scrubbing process were obtained using the methodology as discussed in chapter three.

Table 4.7: Time and Energy Consumption Comparison for 1 Module Fault Injection.

			FLR		Improved FLR	
Fault Injection Matrix	**Number of Cells**	**Number of Frames**	**Error Correction Time (µs)**	**Energy Consumption (nJ)**	**Error Correction Time (µs)**	**Energy Consumption (nJ)**
12,6	42	13	3.565	62.784	2.444	24.441
24,12	84	26	4.286	75.341	2.933	29.329
36,18	126	39	5.000	87.900	3.422	34.218
48,24	168	52	5.714	100.456	3.911	39.106
60,30	210	65	7.811	138.128	5.377	53.771
72,36	252	78	10.632	189.708	7.319	73.185
84,42	294	91	11.778	208.269	8.108	81.076
96,48	336	104	13.492	238.587	9.288	92.878
108,54	378	117	17.042	301.374	11.732	117.320
120,60	420	130	19.885	351.595	13.689	136.870

Figures 4.13 and 4.14 shows the comparison in terms of error correction time against SEU and energy consumption between improved FLR and FLR. The two figures were plotted from Table 4.7 using MATLAB simulation environment.

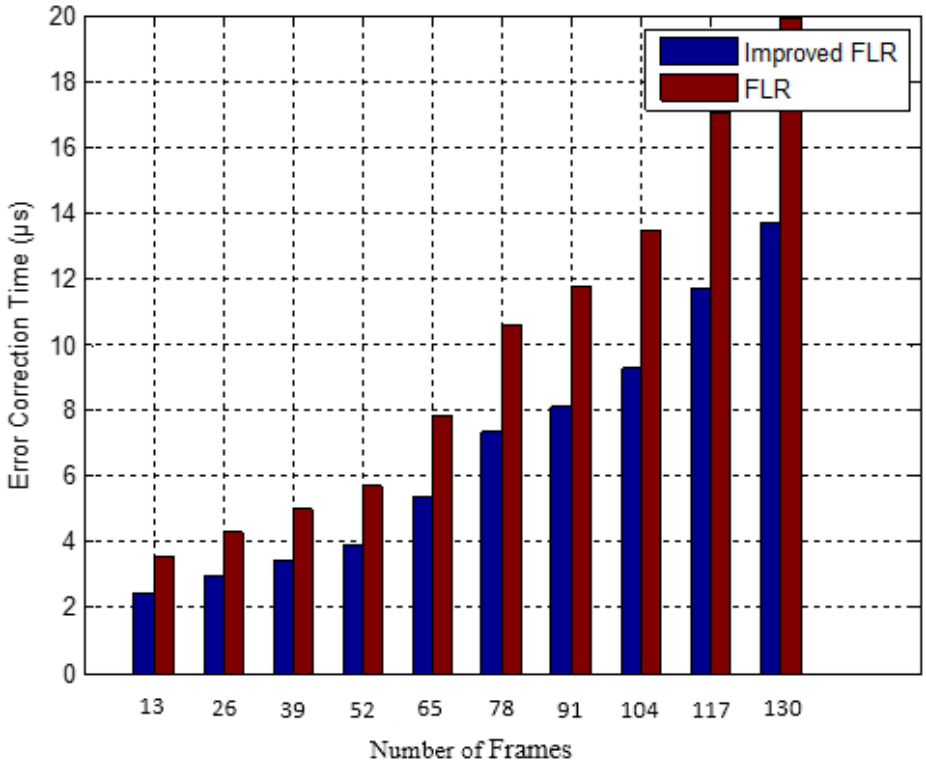


Figure 4.13: Comparison of Error Correction time versus Number of frames in a Module for 1 Module Fault Injection between Improved FLR and FLR.

In Figure 4.13, 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 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 Tonfat *et al.,* 2015) while the FLR scrubbing algorithm took 5 micro seconds. The percentage improvement between Improved FLR and FLR was calculated to be 31.6% using Equation (4.1).

$$\text{Percentage improvement} = \frac{\text{FLR - Improve FLR}}{\text{FLR}} * 100 \quad (4.1)$$

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.

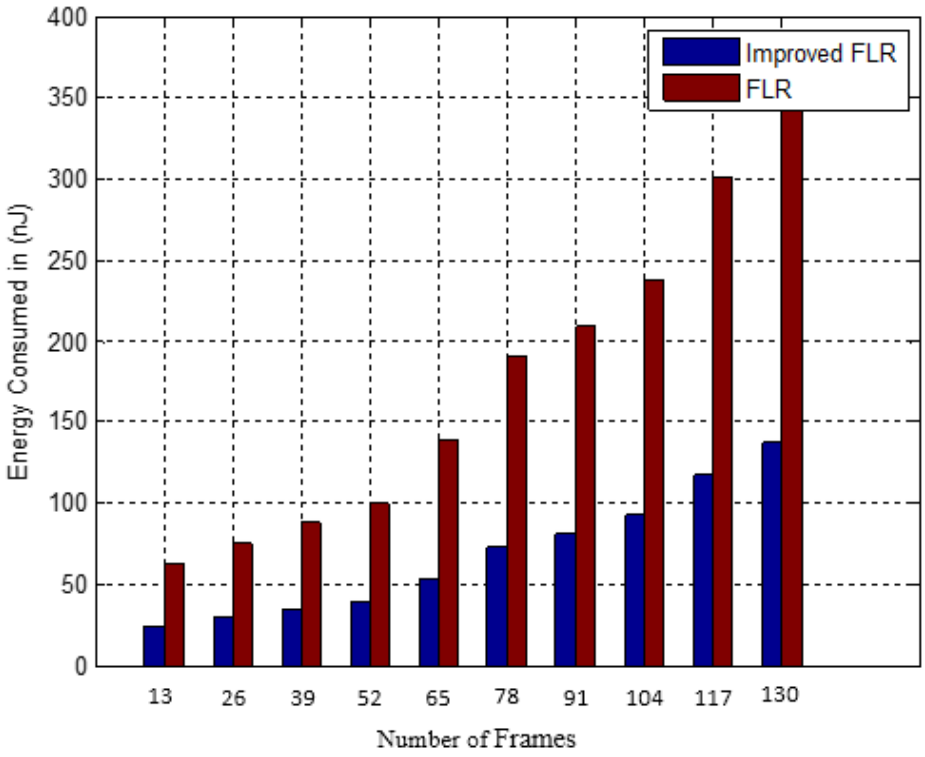


Figure 4.14 Comparison of Energy Consumption versus Number of frames in a Module for 1 Module Fault Injection between Improved FLR and FLR.

Figure 4.14 shows the result for comparison in terms of energy consumption between Improved FLR and FLR scrubbing algorithm. From Figure 4.14, it was 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 was calculated to be 61.1%

using equation (4.1). Significant improvement was also achieved when other module size was examine as seen in Figure 4.14.

CHAPTER FIVE

CONCLUSION AND RECOMMENDATION

5.1 Introduction

This research has developed an improved FLR scrubbing algorithm for static random access memory based field programmable gate array. The developed algorithm is aimed at mitigating single event upset in FPGA configuration memory. The technique introduces CRC as an error detection scheme against SEU. The developed algorithm is evaluated using error correction time against SEU and energy consumption. This chapter summarizes the whole dissertation. Conclusion, significant contributions, and limitations encountered during the course of this research are presented. Recommendations for further research are also suggested.

5.2 Conclusion

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 energetic 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 detected and 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 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 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.

5.3 Significant Contributions

Several researches have been carried out to mitigate SEU in FPGA configuration memory. Most researchers have focused on developing a technique to mitigate SEU without considering the effect of latency. This time delay could cause a negative effect on the functionality of the device. The significant contributions of this dissertation are as follows:

A. Development of an FLR scrubbing algorithm to mitigate SEU in SRAM-based FPGA using CRC as an error detection technique achieved an improvement of 31.6% in error correction time.
B. The improved FLR scrubbing algorithm was also validated using energy consumption. The result showed that the improved FLR algorithm outperformed the FLR scrubbing algorithm by 61.1%.

5.4 Limitation

The limitation of this research work is:

Some built-in logic resources and circuitry could not be scrubbed in Virtex FPGAs because they were not accessible.

5.5 Recommendations for Further Work

The following possible areas of further work are recommended for consideration for future research:

A. SEU mitigation techniques in SRAM-based FPGAs should extend to the application layer as error in some logic resources may propagate to the application layer without being detected by the configuration memory readback.

B. Reliability analysis should be performed on the device under investigation by incorporating a fault-tolerant voter to eliminate the possibility of single point failure.

REFERENCES

Aishwarya, S., & Mahendran, G. (2016). *Multiple Bit Upset correction in SRAM based FPGA using self repairable Erasure codes.* Paper presented at the International Conference on Emerging Engineering Trends and science (ICEETS-2016), 356-362.

Akagic, A., & Amano, H. (2012). *Performance analysis of fully-adaptable CRC accelerators on an FPGA.* Paper presented at the 2012 22nd International Conference on Field Programmable Logic and Applications (FPL), 575-578.

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.

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.

Alagoz, B. B. (2009). Hierarchical Triple-Modular Redundancy (H-TMR) Network For Digital Systems. *arXiv preprint arXiv:0902.0241*, 1-15.

Asadi, H., Tahoori, M. B., Mullins, B., Kaeli, D., & Granlund, K. (2007). Soft error susceptibility analysis of SRAM-based FPGAs in high-performance information systems. *IEEE Transaction on Nuclear Science, 54*(6), 2714-2726.

Badodekar, N. (2009). Mitigating Single-Event Upsets Using Cypress's 65-nm Asynchronous SRAM. *IEEE Transactions on Nuclear Science, 56*(2), 474-478.

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

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.

Bolchini, C., Miele, A., & Sandionigi, C. (2011). A novel design methodology for implementing reliability-aware systems on SRAM-based FPGAs. *IEEE Transactions on Computers, 60*(12), 1744-1758.

Brown, S., & Rose, J. (1996). Architecture of FPGAs and CPLDs: A tutorial. *IEEE Design and Test of Computers, 13*(2), 42-57.

Carmichael, C. H., & Brinkley Jr, P. E. (2006). Techniques for mitigating, detecting and correcting single event upset effects in systems using SRAM-based field programmable gate arrays (pp. 1-26): Google Patents.

Cetin, E., & Diessel, O. (2012). *Guaranteed Fault Recovery Time for FPGA-based TMR Circuits Employing Partial Reconfiguration.* Paper presented at the 2nd International Workshop on Computing in Heterogeneous. Autonomous 'N'Goal-oriented Environments, San Francisco, 1-8.

Chapman, K., & Jones, L. (2010). SEU strategies for Virtex-5 devices. *Xilinx Corporation, 20*, 1-29.

Ciani, L., & Catelani, M. (2014). A fault tolerant architecture to avoid the effects of Single Event Upset (SEU) in avionics applications. *Measurement, 54*, 256-263.

Codinachs, D. M., & Weigand, R. (2009). Overview of FPGA activities in the European Space Agency. *III National School, 23rd April*, 1-55.

Ebrahimi, M., Mohammadi, A., Ejlali, A., & Miremadi, S. G. (2014). A fast, flexible, and easy-to-develop fpga-based fault injection technique. *Microelectronics Reliability, 54*(5), 1000-1008.

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

Faccio, F. (2001). Radiation effects in the electronics for Compact Muon Solenoid, CMS. *The European Organization for Nuclear Research, CERN*, 1-46.

Farooq, U., Marrakchi, Z., & Mehrez, H. (2012). FPGA architectures: An overview *Tree-based Heterogeneous FPGA Architectures* (pp. 7-48): Springer.

Ferguson, R. C., & Tate, R. (2005). *Use of field programmable gate array technology in future space avionics.* Paper presented at the The 24th Conference on Digital Avionics Systems DASC 1-11

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

Habinc, S. (2002). Suitability of reprogrammable FPGAs in space applications. *Gaisler Research," Feasibility Report*, 1-44.

Harding, A., & Wirthlin, M. (2014). *Improving the Reliability of Xilinx 7 Series FPGAs through Configuration Scrubbing.* Paper presented at the International Conference on Field Programmable Logicand Applications, FPL, 1-7.

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.

Heiner, J., Collins, N., & Wirthlin, M. (2008). *Fault tolerant ICAP controller for high-reliable internal scrubbing.* Paper presented at the Aerospace Conference, 2008 IEEE, 1-10.

Heiner, J., Sellers, B., Wirthlin, M., & Kalb, J. (2009). *FPGA partial reconfiguration via configuration scrubbing.* Paper presented at the International Conference on Field Programmable Logic and Applications, FPL2009. , 99-104.

Herrera-Alzu, I., & Lopez, V. M. (2013). Design techniques for Xilinx Virtex FPGA configuration memory scrubbers. *IEEE Transactions on Nuclear Science, 60*(1), 376-385.

Herrera-Alzu, I., & Vallejo, M. L. (2011). Self-reference scrubber for TMR systems based on xilinx virtex FPGAs. *Integrated Circuit and System Design. Power and Timing Modeling, Optimization, and Simulation*, 133-142.

Ibrahim, M. M., Asami, K., & Cho, M. (2013). *Reconfigurable fault tolerant avionics system.* Paper presented at the Aerospace Conference, 2013 IEEE, 1-12.

Ishibashi, K., & Osada, K. (2011). *Low power and reliable SRAM memory cell and array design* (Vol. 31): Springer Science & Business Media.

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.

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.

Johnson, J., & Wirthlin, M. (2009). Voter Insertion Techniques for Fault Tolerant FPGA Design. *I/UCRC Program of the National Science Foundation under Grant No. 0801876*, 1-8.

Johnson, J. M., & Wirthlin, M. J. (2010). *Voter insertion algorithms for FPGA designs using triple modular redundancy.* Paper presented at the Proceedings of the 18th annual ACM/SIGDA international symposium on Field programmable gate arrays, 249-258.

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.

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.

Kastensmidt, F., & Rech, P. (2015). *FPGAs and Parallel Architectures for Aerospace Applications: Soft Errors and Fault-Tolerant Design*: Springer.

Kastensmidt, F., & Rech, P. (2016). FPGAs and Parallel Architectures for Aerospace Applications. *Springer International Publishing Switzerland*, 61-74.

Kastensmidt, F. L., & Carro, L. (2006). Improving reliability of SRAM-based FPGAs by inserting redundant routing. *EEE Transactions on Nuclear Science, 53*(4), 2060-2068.

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.

Li, Y. (2012). *Reliability Techniques for Data Communication and Storage in FPGA-Based Circuits.* (Doctor of Philosophy Doctoral Dissertation), Brigham Young University, Brigham. Retrieved from

http://scholarsarchive.byu.edu/cgi/viewcontent.cgi?article=4500&context=etd, on 28th June, 2016.

Maurer, R. H., Fraeman, M. E., Martin, M. N., & Roth, D. R. (2008). Harsh Environments: Space Radiation Environment, Effects, and Mitigation. *Johns Hopkins APL technical digest, 28*(1), 1-13.

Miller, G., Carmichael, C., & Swift, G. (2007). Single-Event Upset Mitigation for Xilinx FPGA Block Memories. *XILINX Application Note, Virtex-II FPGAs, 1*, 1-19.

Monmasson, E., Idkhajine, L., Cirstea, M. N., Bahri, I., Tisan, A., & Naouar, M. W. (2011). FPGAs in industrial control applications. *IEEE Transactions on Industrial Informatics, 7*(2), 224-243.

Nazar, G. L. (2015). Improving FPGA repair under real-time constraints. *Microelectronics Reliability, 55*(7), 1109-1119.

Nazar, G. L., & Carro, L. (2012). *Exploiting modified placement and hardwired resources to provide high reliability in FPGAs.* Paper presented at the IEEE 20th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), 149-152.

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.

Nigam, A. K., Singh, S., & Tiwari, A. (2015). 6T SRAM Cell: Design And Analysis. *Intl J Engg Sci Adv Research, 1*(3), 125-127.

Ottavi, S. P. M., Cardarilli, V. V. G. C., & Salsano, F. L. A. (2008). Analysis and Evaluations of Reliability of Reconfigurable FPGAs. *Springer Science + Business Media, LLC*, 1-12.

Ozge, A. (2010). *Space radiation Environment and Radiation Hardness Assurance Tests of Electronic Components to be used in Space Missions.* (Doctor of Philosophy Thesis), Middle East Technical University, Anakara. Retrieved from https://etd.lib.metu.edu.tr/upload/12612238/index.pdf, Retrived on 17th June, 2016.

Ranta, J. (2012) The current state of FPGA technology in the nuclear domain. *10* (pp. 1-62): VTT Technical Research Centre of Finland.

Rao, P., Ebrahimi, M., Seyyedi, R., & Tahoori, M. B. (2014). *Protecting SRAM-based FPGAs against multiple bit upsets using erasure codes.* Paper presented at the IEEE Design Automation Conference (DAC), 2014 51st ACM/EDAC, 1-6.

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.

Samudrala, P. K., Ramos, J., & Katkoori, S. (2004). Selective triple modular redundancy (STMR) based single-event upset (SEU) tolerant synthesis for FPGAs. *IEEE Transactions on Nuclear Science, 51*(5), 2957-2969.

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.

Siegle, F., Vladimirova, T., Ilstad, J., & Emam, O. (2015). Mitigation of radiation effects in SRAM-based FPGAs for space applications. *ACM Computing Surveys (CSUR), 47*(2), 1-34.

Sterpone, L., Reorda, M. S., & Violante, M. (2005). *RoRA: a reliability-oriented place and route algorithm for SRAM-based FPGAs.* Paper presented at the 2005 PhD Research in Microelectronics and Electronics, Torino, Italy, 173-176.

Stott, E., Sedcole, P., & Cheung, P. Y. (2008). *Fault tolerant methods for reliability in FPGAs.* Paper presented at the FPL 2008 International Conference on Field Programmable Logic and Applications, London, 415-420.

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.

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

Tarrillo, J., & Lima, K. F. (2014). *Estimating power consumption of multiple modular redundant designs in SRAM-based FPGAs for high dependable applications.* Paper

presented at the 2014 24th International Workshop on Power and Timing Modeling, Optimization and Simulation (PATMOS), 1-7.

Tarrillo, J., Tonfat, J., Tambara, L., Lima Kastensmidt, F., & Reis, R. (2015). *Multiple fault injection platform for SRAM-based FPGA based on ground-level radiation experiments.* Paper presented at the 2015 16th Latin-American Test Symposium (LATS), 1-6.

Tiwari, A., & Tomko, K. A. (2005). Enhanced Reliability of Finite-State Machines in FPGA Through Efficient Fault Detection and Correction. *IEEE Transactions on Reliability, 54*(3), 1-9.

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

Tonfat, J., Tarrillo, J., Tambara, L., Kastensmidt, F. L., & Reis, R. (2016). Multiple Fault Injection Platform for SRAM-Based FPGA Based on Ground-Level Radiation Experiments. *FPGAs and Parallel Architectures for Aerospace Applications*, 135-151.

Vera, G. A., Ardalan, S., Yao, X., & Avery, K. (2013). Fast Local Scrubbing for FPGA's Configuration Memory. *Aerospace Research Central*, 1-21.

Virtanen, A. (2013). Facilities and Radiation Test Methods. *Proceedings of 10th International Workshop on Radiation Effects on Semiconductor Devices for Space Applications*, 1-6.

Wahle, P. J., Schrimpf, R. D., & Galloway, K. F. (1990). Simulated space radiation effects on power MOSFETs in switching power supplies. *IEEE Transactions on Industry Applications, 26*(4), 798-802.

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.

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

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

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.

Yang, L.-Y., Chang, F.-M., Yang, C.-T., & Wang, P.-W. (2011). Static random access memory (SRAM) cell and method for forming same: Google Patents.

Zhang, T., & Ding, Q. (2011). *Design and Implementation of CRC Based on FPGA.* Paper presented at the 2011 Second International Conference on Innovations in Bio-inspired Computing and Applications (IBICA), 160-162.

APPENDIX A

MATLAB m-File "configuration memory generation"

```
function FPGAS
%  Program Description
%%%% N: is the FPGA Row Size
%%%% D: is the FPGA Column Size
%%%% MODUL1: is the first FPGA module;
%%%% MODUL2: is the second FPGA module
%%%% MODUL3: is the third FPGA module
%%%% M: is use to select which module to inject fault
%%%% er: is the fault matrix row
%%%% ed: is the fault matrix column
%%%% n1 and n2: are use to indicate the FPGA row position to
inject fualt
%%%% d1 and d2: are use to indicate the FPGA column position to
inject fualt
clc
close all
N=input('Enter the FPGA row Size:= ');
clc
D=input('Enter the FPGA column Size:= ');
clc
I=input('Specify the Voltage Rating of the FPGA:= ');
clc
V=input('SPecify the Current Rating of the FPGA:= ');
clc
MODUL1=rand(N,D)>0.8;
MODUL2=MODUL1;
MODUL3=MODUL2;
% ERROR=rand(1)>0.8;
M=input('ENTER 1 TO INJECT FAULT IN MODULE 1:\nENTER 2 TO INJECT
FAULT IN MODULE 2:\nENTER 3 TO INJECT FAULT IN MODULE 3:\nENTER 12
TO INJECT FAULT IN MODULE 1 and 2:\nENTER 13 TO INJECT FAULT IN
MODULE 1 and 3:\nENTER 23 TO INJECT FAULT IN MODULE 2 and
3:\nENTER 123 TO INJECT FAULT IN MODULE 1,2 and 3:\n\n:= ');
disp('')
disp('')
er=input('Please Enter the Fault Matrix Row:= ');
ed=input('Please Enter the Fault  Matrix Colomn:= ');
n1=input('Please Indicate the FPGA Starting Row of Error:= ');
n2=input('Please Indicate the FPGA Ending Row of Error:= ');
d1=input('Please Indicate the FPGA Starting Coloumn of Error:= ');
d2=input('Please Indicate the FPGA Ending Coloumn of Error:= ');
```

APPENDIX B

MATLAB m-File "Fault Injection"

```
%M=1 correspond to fault injected in module 1
ERROR=rand(er,ed)>0.8;
if M==1
    [m,n]=size(MODUL1);
%     for c = 1:n
%     MODUL1(randperm(m),c) = MODUL1(:,c);
%     end
%     mFAULT=MODUL1
cols = randperm(n);
k = n;  %//This is a parameter you'll need to tweak...
set_size =n/k;% floor(n/k);
for set = 1:set_size:(n/2)
    set_cols = cols(set:(set+set_size-1));
%     MODUL1(:,set_cols) = MODUL1(randperm(m), set_cols);
end
MODUL1(n1:n2,d1:d2)=ERROR;
mFAULT=MODUL1
MODUL2
MODUL3
end
%M=2 correspond to fault injected in module 2
if M==2
    [m,n]=size(MODUL2);
%     for c = 1:n
%     MODUL2(randperm(m),c) = MODUL2(:,c);
%     end
%     mFAULT=MODUL2

cols = randperm(n);
k = 5;  %//This is a parameter you'll need to tweak...
set_size =n/k;% floor(n/k);
for set = 1:set_size:n
    set_cols = cols(set:(set+set_size-1));
%     MODUL2(:,set_cols) = MODUL2(randperm(m), set_cols);
end
MODUL1
MODUL2(n1:n2,d1:d2)=ERROR;
mFAULT=MODUL2
MODUL3
end

%M=3 correspond to fault injected in module 3
if M==3
    [m,n]=size(MODUL3);
%     for c = 1:n
%     MODUL3(randperm(m),c) = MODUL3(:,c);
%     end
```

```
cols = randperm(n);
k = 5;  %//This is a parameter you'll need to tweak...
set_size =n/k;% floor(n/k);
for set = 1:set_size:n
    set_cols = cols(set:(set+set_size-1));
%     MODUL3(:,set_cols) = MODUL3(randperm(m), set_cols);
end
MODUL1
MODUL2
MODUL3(n1:n2,d1:d2)=ERROR;
mFAULT=MODUL3
end

%%%%%%%%%%%%%%%%%INJECT OTHER FAULTS HERE%%%%%%%%%%%%%%
%Inject fault in module 1 and 2 and leave keep module 3
if M==12
    [m,n]=size(MODUL1);
    [m,n]=size(MODUL2);
    cols = randperm(n);
k = 5;  %//This is a parameter you'll need to tweak...
set_size =n/k;% floor(n/k);
    for set = 1:set_size:n
    set_cols = cols(set:(set+set_size-1));
%     MODUL1(:,set_cols) = MODUL1(randperm(m), set_cols);
%     MODUL2(:,set_cols) = MODUL2(randperm(m), set_cols);
    end

MODUL1(n1:n2,d1:d2)=ERROR;
ERROR=rand(er,ed)>0.8;
MODUL2(n1:n2,d1:d2)=ERROR;
mFAULT1=MODUL1
mFAULT2=MODUL2
MODUL3
end
%Inject fault in module 1 and 3 and leave keep module 2
if M==13
    [m,n]=size(MODUL1);
    [m,n]=size(MODUL3);
    cols = randperm(n);
k = 5;  %//This is a parameter you'll need to tweak...
set_size =n/k;% floor(n/k);
    for set = 1:set_size:n
    set_cols = cols(set:(set+set_size-1));
%     MODUL1(:,set_cols) = MODUL1(randperm(m), set_cols);
%     MODUL3(:,set_cols) = MODUL3(randperm(m), set_cols);
    end
MODUL1(n1:n2,d1:d2)=ERROR;
ERROR=rand(er,ed)>0.8;
MODUL2(n1:n2,d1:d2)=ERROR;
mFAULT1=MODUL1
MODUL2
mFAULT3=MODUL3
```

```
end
%Inject fault in module 2 and 3 and leave keep module 1
if M==23
    [m,n]=size(MODUL3);
    [m,n]=size(MODUL2);
    cols = randperm(n);
k = 5;  %//This is a parameter you'll need to tweak...
set_size =n/k;% floor(n/k);
    for set = 1:set_size:n
    set_cols = cols(set:(set+set_size-1));
%     MODUL3(:,set_cols) = MODUL3(randperm(m), set_cols);
%     MODUL2(:,set_cols) = MODUL2(randperm(m), set_cols);
    end
MODUL1
MODUL2(n1:n2,d1:d2)=ERROR;
ERROR=rand(er,ed)>0.8;
MODUL3(n1:n2,d1:d2)=ERROR;
mFAULT2=MODUL1
mFAULT3=MODUL3
end
%Inject fault in module 1 and 2 and leave keep module 3
if M==123
    [m,n]=size(MODUL1);
    [m,n]=size(MODUL2);
    [m,n]=size(MODUL3);
    cols = randperm(n);
k = 5;  %//This is a parameter you'll need to tweak...
set_size =n/k;% floor(n/k);
    for set = 1:set_size:n
    set_cols = cols(set:(set+set_size-1));
%     MODUL1(:,set_cols) = MODUL1(randperm(m), set_cols);
%     MODUL2(:,set_cols) = MODUL2(randperm(m), set_cols);
%     MODUL3(:,set_cols) = MODUL3(randperm(m), set_cols);
    end
MODUL1(n1:n2,d1:d2)=ERROR;
ERROR=rand(er,ed)>0.8;
MODUL2(n1:n2,d1:d2)=ERROR;
ERROR=rand(er,ed)>0.8;
MODUL3(n1:n2,d1:d2)=ERROR;
mFAULT1=MODUL1
mFAULT2=MODUL2
mFAULT3=MODUL3
end
disp('Press Any Key to Implement Voting')
pause
if M==1
    Voting(MODUL2,MODUL3,mFAULT,I,V)
elseif M==2
    Voting(MODUL1,MODUL3,mFAULT,I,V)
elseif M==3
    Voting(MODUL1,MODUL2,mFAULT,I,V)
end
```

```
if M==12
    Voting(mFAULT1,mFAULT2,MODUL3,I,V)
elseif M==13
    Voting(mFAULT1,MODUL2,mFAULT3,I,V)
elseif M==23
    Voting(MODUL1,mFAULT2,mFAULT3,I,V)
end
if M==123
    Voted_Module3=Voting(mFAULT1,mFAULT2,mFAULT3,I,V)
    Original_Bits=MODUL1
End
```

APPENDIX C

MATLAB m-File "cyclic redundancy check"

```
function OUT=CRC(A,B,C,D)
 [r,c]=size(A);
for i=1:r
    for j=1:c
        if A(1:end,1)==B(1:end,1)
            if A(1:end,1)==C(1:end,1)
                if B(1:end,1)==C(1:end,1)
                    if D(1:end,1)==A(1:end,1)
                        if D(1:end,1)==B(1:end,1)
                            if D(1:end,1)==C(1:end,1)
                                ww=A
                                XX=B;
                                YY=C;
                                ZZ=D;
                            end
                        end
                    end
                end
            end
        end
    end
end
```

APPENDIX D

MATLAB m-File "bit-level Voting"

```
%PERFORM VOTING HERE
% indMF=find(mFAULT==1);
% indMF0=find(mFAULT==0);
% ind1=find(MODUL1==1);%Determine the 1 Element of MODULE one
% ind10=find(MODUL1==0);%Determine the 0 Element of MODULE one
% ind2=find(MODUL2==1);%Determine the 1 Element of MODULE two
% ind20=find(MODUL2==0);%Determine the 0 Element of MODULE two
% ind3=find(MODUL3==1);%Determine the 1 Element of MODULE three
% ind30=find(MODUL3==0);%Determine the 0 Element of MODULE three
% [r1,c1]=ind2sub(size(MODUL1),ind1);
% [r10,c10]=ind2sub(size(MODUL1),ind10);
% [r2,c2]=ind2sub(size(MODUL2),ind2);
% [r20,c20]=ind2sub(size(MODUL2),ind20);
% [r3,c3]=ind2sub(size(MODUL3),ind3);
% [r30,c30]=ind2sub(size(MODUL3),ind30);
% [rf,cf]=ind2sub(size(mFAULT),indMF);
% [rf0,cf0]=ind2sub(size(mFAULT),indMF0);
%
% for j=1:length(cf)
%      if ind1(j,:)==(ind2(j,:)) & (ind1(j,:))~=(indMF(j,:))
%         mFAULT(rf(j),cf(j))=1;
%      elseif ind10(j,:)==ind2(j,:) && ind10(j,:)~=indMF0(j)
%         mFAULT(rf(j,:),cf(j,:))=0
%      end
% %      A(j,:)=mFAULT(rf(j,:),cf(j,:));
% end


function [Voted_Module1,Voted_Module2,Voted_Module3,
E]=Voting(A,B,C,I,J)

[n,m]=size(A);
Original=A;
for i=1:n
    for j=1:m
        tic
        if A(i,j)==B(i,j) & A(i,j)~=C(i,j)
        T_Frame=toc;
            C(i,j)=A(i,j);
            Voted_Module=C
        end
        tic
        if A(i,j)~=C(i,j)&B(i,j)~= C(i,j)
        T_Frame=toc;

            A(i,j)=C(i,j);
```

```
            B(i,j)=C(i,j);
            Voted_Module1=A
            Voted_Module2=B
        end
        T_Frame=toc;
        if B(i,j)~=A(i,j)&C(i,j)~=A(i,j)
        T_Frame=toc;
            B(i,j)=A(i,j);
            C(i,j)=A(i,j);
            Voted_Module1=B
            Voted_Module2=C
        end
        T_Frame=toc;
        if A(i,j)~=B(i,j)&C(i,j)~=B(i,j)
        T_Frame=toc;
            A(i,j)=B(i,j);
            C(i,j)=B(i,j);
            Voted_Module1=A
            Voted_Module2=C
        end
        T_Frame=toc;
        if A(i,j) ~= Original(i,j);
        T_Frame=toc;
            if B(i,j)~=Original(i,j)
                if C(i,j)~=Original(i,j)
                    C(i,j)=Original(i,j);
                    B(i,j)=Original(i,j);
                    A(i,j)=Original(i,j);
                    Voted_Module1= A%(i,j)
                    Voted_Module2= B%(i,j)
                    Voted_Module3= C;%(i,j)
                 end
            end
        end
    end
end
E=I*J*T_Frame
```