A Method to Estimate the Reliability of FPGA-based Architectures for Space Systems

by

Juan Andrés Pérez Celis

A thesis submitted to the Computer Science Department, in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN THE AREA OF
COMPUTER SCIENCE

at the

Instituto Nacional de Astrofísica, Óptica y Electrónica
Tonantzintla, Puebla
August 2016

Advisor:

PhD René Cumplido Parra
Professor at INAOE

© INAOE 2016
All rights reserved
The author hereby grants to INAOE permission to reproduce and to distribute paper or electronic copies of the thesis document in whole or in part
Acknowledgments

This work could not have been possible without the continuous support of my advisor, Dr. René Cumplido. He gave me an excellent research opportunity and guided me to overcome each obstacle that we encountered. Dra. Claudia Feregrino also made possible the interaction with BYU and MIT, thank you for your support and believing in me. Dr. Michael Wirthlin and Dr. Alvar Saenz-Otero were also instrumental in providing clarity to the work.

I wish to thank the members of the Configurable Lab at BYU for their valuable time, patience, ideas to cope with the problems we faced with PicoBlaze and the opportunity to perform a neutron test. The work would not be possible without you Andrew, Ammon, Chase, Dr. Wirthlin, and all of the members who helped me on the way.

I am very grateful to my parents, my grandma, my brothers and my girlfriend who fueled me to overcome any challenges life presents and to reach any goals I set. This milestone would not be possible without your help, without your advice, and without your patience. I will always be grateful to all of you.

This work was supported by MSc scholarship No 392068 by CONACyT and under CONACyT grant PDCPN2013-01-216689.
Abstract

The use of Static Random Access Memory (SRAM)-based Field Programmable Gate Arrays (FPGAs) in space systems is gaining a growing interest. The high-computational power, the ability to reconfigure and the flexibility of FPGAs at low power make them suitable devices for space applications. Nevertheless, FPGAs are susceptible to Single Event Upsets which may cause an operational failure, leading to a decrease of the reliability. Fault Mitigation Techniques (FMT) are used to increase the reliability, making the architecture suitable for operating in the space. However, estimating the reliability of a FPGA design can be challenging. The preferred method consists of placing the FPGA under a high-energy particle beam and performing the test iteratively to estimate the reliability. The iterative process makes radiation testing prohibitive for many space applications, as the monetary cost and time spent for radiation testing is extremely high.

This thesis proposes a method for determining the reliability of an FPGA architecture with FMTs applied based on the failure rate of an unmitigated design. The method uses Markov chains to model the architecture and estimate the reliability. An unmitigated design and a mitigated design were developed and placed under a neutron beam to estimate the reliability. The mitigated design decreases the sensible area, i.e. increases the reliability, by a factor of 20.18. The mitigated design is a triplication of the mitigated design with additional voters placed throughout the architecture. Two models resembling the behavior of the mitigated design with a Medium Grain Triple Modular Redundancy (MGTMR) are proposed. The difference of the models lies in the addition of shared-resources modules. The results of the models show that the estimation of reliability is close to that obtained through neutron testing.

Keywords: Space Systems, FPGAs, Reliability, Markov Chains, Fault Mitigation Techniques, Voters Placement.
# CONTENTS

List of Tables XI

List of Figures XIII

Acronyms XV

1 Introduction 1
   1.1 Problem Statement . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4
   1.2 Hypothesis . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5
   1.3 Justification . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5
   1.4 Objectives . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6
      1.4.1 General Objective . . . . . . . . . . . . . . . . . . . . . . . . . . . 6
      1.4.2 Specific Objectives . . . . . . . . . . . . . . . . . . . . . . . . . . 7
   1.5 Contribution . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 7
   1.6 Structure . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 8

2 Background Information 9
2.1 FPGA Architecture ................................................. 9
  2.1.1 Xilinx 7-series Architecture ................................. 11
  2.1.2 Reconfiguration ............................................... 13
2.2 Radiation Overview .............................................. 16
  2.2.1 Radiation Environment ....................................... 16
  2.2.2 Radiation Effects on FPGAs ................................. 17
2.3 FPGA Fault Mitigation Techniques ............................... 26
  2.3.1 Reliability Overview ........................................ 26
  2.3.2 Fault Mitigation ............................................. 27
2.4 FPGA’s Architecture Testing ..................................... 28

3 Related Work ..................................................... 31
  3.1 Markov Modeling ................................................ 31
  3.2 Reliability Challenges .......................................... 33
  3.3 Voters Placement .............................................. 39
  3.4 Remarks ........................................................ 40

4 Reliability Model ................................................ 41
  4.1 Proposed Method .............................................. 42
  4.2 Proposed Models .............................................. 45
    4.2.1 First Model ............................................. 45
    4.2.2 Second Model ............................................. 48

5 Architecture under Test ..................................... 55
  5.1 Vanilla Design .............................................. 57
  5.2 Medium Grain TMR ........................................... 60
  5.3 BLTMR Design ............................................... 62
  5.4 Experimental Setup .......................................... 63

6 Results ......................................................... 67
  6.1 Models Evaluation ........................................... 67

— VI —
LIST OF TABLES

2.1 FMTs for elements in SRAM FPGAs[37]. ....................................... 29

3.1 Testing Methods. ................................................................. 38

5.1 Fault injection results .......................................................... 64

6.1 Results of Vanilla neutron testing. ........................................... 68
6.2 Results of MGTMR neutron testing. ......................................... 69
6.3 Summarized results of Vanilla and MGTMR. .............................. 69
6.4 Results of BLTMR neutron testing. ......................................... 72
6.5 Summarized results of Vanilla, MGTMR and BLTMR. ................. 73
## LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>CMOS SRAM cell.</td>
<td>11</td>
</tr>
<tr>
<td>2.2</td>
<td>Tiled CLBs [53].</td>
<td>12</td>
</tr>
<tr>
<td>2.3</td>
<td>Xilinx FPGA Layout [15].</td>
<td>13</td>
</tr>
<tr>
<td>2.4</td>
<td>Partial reconfiguration functionality [26].</td>
<td>14</td>
</tr>
<tr>
<td>2.5</td>
<td>Frames over FPGA layout [27].</td>
<td>15</td>
</tr>
<tr>
<td>2.6</td>
<td>Charged particles in space environment [56].</td>
<td>18</td>
</tr>
<tr>
<td>2.7</td>
<td>Flux on a target area [59].</td>
<td>19</td>
</tr>
<tr>
<td>2.8</td>
<td>Particle actual path [59].</td>
<td>20</td>
</tr>
<tr>
<td>2.9</td>
<td>Funneling effect in CMOS transistor [59].</td>
<td>22</td>
</tr>
<tr>
<td>2.10</td>
<td>Charged particles in Van Allen belts [56].</td>
<td>25</td>
</tr>
<tr>
<td>4.1</td>
<td>Block diagram of the main steps to estimate the reliability.</td>
<td>43</td>
</tr>
<tr>
<td>4.2</td>
<td>Modules and portions for first approach.</td>
<td>46</td>
</tr>
<tr>
<td>4.3</td>
<td>Markov chain for the first approach.</td>
<td>46</td>
</tr>
<tr>
<td>4.4</td>
<td>Model 1 reliability curve.</td>
<td>48</td>
</tr>
<tr>
<td>Section</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>---------</td>
<td>------------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>4.5</td>
<td>Second model portion division.</td>
<td>49</td>
</tr>
<tr>
<td>4.6</td>
<td>States’ description for the second model.</td>
<td>50</td>
</tr>
<tr>
<td>4.7</td>
<td>Markov chain of the MGTMR architecture.</td>
<td>51</td>
</tr>
<tr>
<td>4.8</td>
<td>Model 2 reliability curve.</td>
<td>53</td>
</tr>
<tr>
<td>5.1</td>
<td>PB softprocessor block diagram.</td>
<td>56</td>
</tr>
<tr>
<td>5.2</td>
<td>Block diagram of BRAM scrubbers.</td>
<td>58</td>
</tr>
<tr>
<td>5.3</td>
<td>Picoblaze vanilla layout.</td>
<td>59</td>
</tr>
<tr>
<td>5.4</td>
<td>Proposed failure detector architecture.</td>
<td>59</td>
</tr>
<tr>
<td>5.5</td>
<td>Medium grain architecture diagram [52].</td>
<td>62</td>
</tr>
<tr>
<td>5.6</td>
<td>Picoblaze fault injection histogram (vanilla on the left, MGTMR on the right).</td>
<td>65</td>
</tr>
<tr>
<td>5.7</td>
<td>Picoblaze vanilla fault injection.</td>
<td>65</td>
</tr>
<tr>
<td>6.1</td>
<td>Experimental reliability curve.</td>
<td>70</td>
</tr>
<tr>
<td>6.2</td>
<td>Proposed models and experimental reliability curve.</td>
<td>71</td>
</tr>
</tbody>
</table>
## ACRONYMS

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABFT</td>
<td>Algorithm-Based Fault Tolerance</td>
</tr>
<tr>
<td>BLTMR</td>
<td>BYU-LANL TMR</td>
</tr>
<tr>
<td>BRAM</td>
<td>Block of Random Access Memory</td>
</tr>
<tr>
<td>CEU</td>
<td>Code Emulated Upsets</td>
</tr>
<tr>
<td>CG</td>
<td>Coarse Grain</td>
</tr>
<tr>
<td>DWC</td>
<td>Duplication With Compare</td>
</tr>
<tr>
<td>EDAC</td>
<td>Error Detection And Correction</td>
</tr>
<tr>
<td>FG</td>
<td>Fine Grain</td>
</tr>
<tr>
<td>FMT</td>
<td>Fault Mitigation Techniques</td>
</tr>
<tr>
<td>IOB</td>
<td>Input/Output Blocks</td>
</tr>
<tr>
<td>LANSCE</td>
<td>Los Alamos Neutron Science Center</td>
</tr>
<tr>
<td>LEO</td>
<td>Low Earth Orbit</td>
</tr>
<tr>
<td>LET</td>
<td>Linear Energy Transfer</td>
</tr>
<tr>
<td>LUT</td>
<td>Look-Up Table</td>
</tr>
<tr>
<td>MBU</td>
<td>Multiple-Bit Upset</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Definition</td>
</tr>
<tr>
<td>--------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>MCU</td>
<td>Multiple-Cell Upset</td>
</tr>
<tr>
<td>MG</td>
<td>Medium Grain</td>
</tr>
<tr>
<td>nMR</td>
<td>n-Modular Redundancy</td>
</tr>
<tr>
<td>PB</td>
<td>PicoBlaze</td>
</tr>
<tr>
<td>PROM</td>
<td>Program Read-Only Memory</td>
</tr>
<tr>
<td>RPR</td>
<td>Reduced Precision Redundancy</td>
</tr>
<tr>
<td>RTL</td>
<td>Register Transfer Logic</td>
</tr>
<tr>
<td>SEE</td>
<td>Single Event Effect</td>
</tr>
<tr>
<td>SEFI</td>
<td>Single Event Functional Interrupt</td>
</tr>
<tr>
<td>SEGR</td>
<td>Single Event Gate Rupture</td>
</tr>
<tr>
<td>SEL</td>
<td>Single Event Latch-up</td>
</tr>
<tr>
<td>SET</td>
<td>Single Event Transient</td>
</tr>
<tr>
<td>SEU</td>
<td>Single Event Upset</td>
</tr>
<tr>
<td>TID</td>
<td>Total Ionizing Dose</td>
</tr>
<tr>
<td>TMR</td>
<td>Triple Modular Redundancy</td>
</tr>
<tr>
<td>VD</td>
<td>Vanilla Design</td>
</tr>
</tbody>
</table>
INTRODUCTION

The use of Field Programmable Gate Arrays (FPGAs) in space missions is growing due to the reconfiguration ability, market availability, high performance and low power consumption. FPGAs have been used in several space missions. FPGAs were tested at the International Space Station (ISS) with the experiment SpaceCube \(^1\), which goal was to demonstrate high-intensity computing systems with FPGAs in Low Earth Orbit (LEO) \(^1\), \(^2\). The project stayed on ISS for more than three years. The project was a success enabling the possibility of doing more versions of SpaceCube.

Another example is the use of Xilinx Virtex 4 and Virtex QPro in the Mars Rover, landed by NASA and Jet Propulsion Laboratory (JPL) on August 2012 \(^2\). More missions can be found with FPGAs on them, one of the most recent applications is the Flying

\(^1\)http://www.nasa.gov/mission_pages/station/research/experiments/487.html
\(^2\)http://www.xilinx.com/about/customer-innovation/aerospace-and-defense/mars-exploration-rovers.html
1. INTRODUCTION

Laptop \(^3\) which has the On Board processing system based on only FPGAs [18], [19].

Unfortunately, SRAM-based FPGAs are vulnerable to radiation environments. The radiation of the space weather and the presence of neutrons in Earth cause bit flips, also known as Single Event Upsets (SEUs), in memory elements comprising the FPGA. The presence of SEUs may induce a fault in the FPGA which may become a failure, severely impacting the operation of FPGA system [4]. The bit flip induces a fault if it is presented in resources used by the architecture.

Neutrons can originate the fault in the Earth environment, charged particles in space weather [8], [7] or at high-energy physics experiments [3]. Physically, the effect made by a neutron or a charged particle is different; however, the manifestation of the effect is the same: a bit flip in the FPGA. Bit flips are also known as upsets; if only one bit flips it is called a Single Event Upset (SEU) [4], if more than one-bit flips it is called a Multiple Bit Upset (MBU). The upset may cause a failure or not, depends on where the FPGA is hit. FPGAs are being used for critical applications in high radiation environments event though they may fail. Several aspects are considered when designing with FPGAs to cope with the radiation environment.

It is necessary to take into account the reliability and the availability to use an FPGA in critical applications. Reliability is the probability of performing correctly during a given time.[20] Availability is the probability of being operational in a given time. The R&A are parameters provided as a requirement during the planning phase of a system. In critical applications, to meet the requirements the system should be fault protected. Applying Fault Mitigation Techniques (FMT) grants protection to the FPGA. The techniques correct, mask or detect a fault. Numerous FMT exist, choosing the suitable FMT can be challenging, even when following guidelines such as the ones provided by NASA Goddard Space Center [49], [50]. Moreover, within the most used FMT – the Triple Modular Redundancy – the position where majority voters are placed impacts the reliability of the architecture. An addition of several majority voters does

\(^3\)https://directory.eoportal.org/web/eoportal/satellite-missions/f/flying-laptop
not necessarily mean an improvement of the architecture [32]. However, the greatest challenge is testing the FPGAs to estimate the R&A.

Several types of test exist to estimate the parameters, the most used for FPGAs are: life testing [6], radiation testing [7] [8] [35], fault injection [9] and analytical techniques [14]. The test that yields better results is radiation testing as it simulates the environment where the FPGA will operate. However, using a radiation test facility is expensive. Considering that designing is an iterative process, the cost and time for testing are highly incremented. On the other hand, the most used test is fault injection, where faults are controllably introduced to the design. The faults are bit-flips in several elements of the FPGA. The drawback of this method is that not all FPGA is addressable; i.e., faults can not be injected in all the design. As not all the design is tested the test allows poor estimation of reliability.

The FPGA architecture can be tested in few cases, and the R&A estimated, due to time and budget constraints. The estimation is only performed for expensive and high-impact projects (expensive NASA missions, projects where humans may be harm). Large is the number of cases where assessing the R&A is not feasible, in these cases, only the system’s operation is tested before launching the system [18], [19]. For cases where FPGAs are proposed as an innovative solution within a project, most of the time there is no budget to test the FPGA solution iteratively. The use of the CubeSats [17] – a popular standard of small cube-shaped satellites measuring 10 cm per side, commonly used for testing innovative systems and technologies for their low intrinsic cost– and the case of the ATLAS project [3] can be appended to the list.

In the light of the above, a method to estimate the reliability of an FPGA architectures is needed. This thesis focuses on providing a method for determining the reliability of an FPGA architecture with FMTs applied based on the failure rate of an unmitigated design. The method uses Markov chains to model the architecture and estimate the reliability. After completing the method, medical, automotive, space, high-energy physics and other projects will benefit by enabling the use of FPGA
devices, reducing the time to test their products and saving money of renting testing facilities. Besides, the method will have a positive impact in critical application projects, where the budget is limited or where FPGAs are seen as an innovative solution. Institutions around the world that could not afford a proper test will now be able to estimate the reliability of their architectures. Providing institutions with low-cost testing will help to estimate the R&A in an early stage of the design. Thus, a decision of the feasibility of using an FPGA solution can be taken based on substantial evidence provided by the model results.

1.1 Problem Statement

Meeting all the requirements imposed by the project is mandatory when designing a system for critical applications in radiation environments. Meeting the reliability requirement of the FPGA architecture is not the exception. Nowadays, testing an FPGA architecture is expensive and time-consuming. Furthermore, for several applications, it is not possible to estimate the reliability of the design. Numerous low-budget projects are launched without a reliability test on the FPGA. Even for space agencies not all FPGA architectures can be tested. Likewise, projects can not afford radiation testing when FPGAs are proposed as an innovative solution. It is necessary to have a method that allows testing SRAM FPGA architectures to evaluate the reliability at a low cost, enabling the possibility for testing several architectures without restrictions for industries, agencies, and academic institutions. By using this method substantial money and time savings may be presented when designing FPGA architectures. Moreover, the use of FPGA in critical applications will be further expanded.
1.2 Hypothesis

This thesis seeks to provide evidence on the hypothesis that:

- If the reliability and the total area of an unmitigated FPGA architecture are known, the reliability of the same architecture after applying mitigation techniques can be estimated by analyzing the architecture’s area between added elements of the mitigation technique with Markov chains.

1.3 Justification

In projects involving FPGAs the architectures synthesized are tested in functionality. However, few are the cases where a reliability estimation is performed. Even for critical applications, such as space missions, the evaluation of the reliability is not always performed. Projects with a large budget are more likely to test their architecture in a proton or heavy ion beam iteratively. Projects with a limited budget cannot afford multiple tests under the beam. Moreover, the results given by a dynamic test under the beam are particular to each architecture; i.e., if the architecture is modified another test should be performed. The cost and time of testing iteratively become prohibitive for most space projects.

Alternative tests to cope with the issue can be done. However, those tests are not as promising as the radiation test. The most common test is fault injection [9]. During fault injection, a device performs the task of changing each of the configuration bits of all the FPGA and waits for a fault. If a bit causes a fault it is marked as a sensitive bit. The result is the percentage of sensitive bits out of the complete configuration bitstream. Nonetheless, fault injection is a time-consuming test because after injecting a fault sufficient time should be allowed for the fault to propagate throughout the architecture to determine if it causes a failure. Furthermore, faults cannot be injected
in all the components of an FPGA.

Another type of test which is becoming more popular for neutron testing is life testing [6]. Life testing consists of placing the architecture under the desired environment and acquiring data throughout the years. Xilinx places several banks of FPGAs in different regions with the goal of estimating the reliability of the FPGAs. However, the experiments should run for years to gather enough data to estimate the reliability [6]. Data from several years can be collected in a matter of hours in a heavy ion beam [7].

Spending years testing an architecture is not reasonable in space projects. Affording several beam test is also not plausible for most of the space projects. Aside from iterative beam testing, there is not an alternative to fairly estimate the reliability of an FPGA architecture. This thesis presents a method to estimate the reliability of an architecture with FMTs based on the results of testing under a beam an unmitigated design.

1.4 Objectives

1.4.1 General Objective

The aim of this work is to develop a method to estimate the reliability of the architecture after the addition of mitigation techniques.

The method proposes the use Markov chains to model the reliability of the mitigated design. The inputs to the model are the results of testing the unmitigated design under a beam; namely, the inputs are the failure rate and the sensitive area of the unmitigated architecture. Likewise, the resource utilization of the mitigated architecture provides information to the model to estimate the reliability.
1.4.2 Specific Objectives

The following objectives should be met:

- Synthesize an unmitigated architecture capable of collecting data based on the microprocessor PicoBlaze for further a test under a neutron beam.

- Determine the effect on the reliability of the placement of majority voters within the architecture.

1.5 Contribution

The main contribution of this thesis is the development of a method to estimate the reliability of mitigated designs without actual beam testing them. The approach taken considers the area between the resources added with the FMTs. The method consists of making a Markov chain where the states represent the modules of the architecture; the transactions are the probabilities of failure or recovery of each state. The probabilities are related to the sensitive area of the modules and the overall failure rate of the unmitigated design. The successful representation of the architecture with the proposed model represents evidence supporting that a fair reliability estimation of a given architecture can be made based on data thrown by radiating the unmitigated design and computing the resources comprised between added logic in the mitigated design.

Another contribution of the work is in supporting the impact on the reliability of the placement of majority voters within a Medium Grain Triple Modular Redundancy (MGTMR). Namely, that an increased number of majority voters will not necessarily yield a higher reliability of the architecture.
1.6 Structure

The thesis is divided into 7 Chapters to allow a better understanding of the work performed. The rest of the document is organized as follows:

**Chapter 2:** Provides the information and definitions needed to gain an in-depth understanding of the topics discussed throughout the work.

**Chapter 3:** Presents a brief review of relevant works related to reliability modeling and estimation for electronic devices.

**Chapter 4:** Discusses the proposed method and models for estimating the reliability of an FPGA architecture.

**Chapter 5:** Describes the architectures synthesized for neutron testing. Likewise, the experimental setup is presented.

**Chapter 6:** Shows the experimental results of radiating the architectures with a neutron beam.

**Chapter 7:** Presents the conclusions of the work and provides a guide for future work.
This chapter is comprised of useful information to understand the work performed in this thesis. Section 2.1 describes the FPGAs Architecture with particular emphasis on the 7-series of Xilinx. Section 2.2 gives an explanation of the radiation environment and its effect on FPGAs. Section 2.3 shows several fault mitigation techniques that are currently used on FPGAs. Section 2.4 describes the testing options for FPGA architectures.

2.1 FPGA Architecture

The Field Programmable Gate Array (FPGA) is a programmable logic device (PLD). The first models were introduced back in the 1980s. In FPGAs, the user can synthesize an architecture using a hardware description language. There are three types of FPGAs:
2. BACKGROUND INFORMATION

- Antifuse.
- Flash.
- Static Random Access Memory (SRAM).

The antifuse FPGA consists of a one-time programmable electrical configuration memory. The desired logic is realized by burning off contact points in the circuit. They are non-volatile. Even though antifuse FPGAs were preferred in the 1980s for the stability of their configuration memory, and are still being used in critical applications, most of the stability problems for volatile FPGAs were solved enabling a broad range of applications for the recent Reconfigurable Computing (RC) area.

The Flash FPGAs are nonvolatile and reconfigurable. The configuration memory is stored in a flash memory. They are less power consumptive and are also more tolerant to radiation effects. Despite the fact that FLASH FPGAs can pack more than 150,000 logic elements, the number is only comparable to a small size SRAM FPGA.

SRAM FPGAs are reconfigurable. The logic behavior and routing are stored into an SRAM memory [15]. The technology is volatile, and the program should be loaded after powering the FPGA up. SRAM FPGAs can be reprogrammed as many times as needed with different bitstreams, representing a suitable technology for critical projects. During this thesis, all references to FPGAs are for SRAM-based FPGAs, unless otherwise stated.

All types of FPGAs are becoming more valuable for space applications because of their high density, high performance, reduced development cost, and re-programmability. In particular, SRAM-based FPGAs are very valuable because of the possibility of being reprogrammed by the user and the higher density when compared to others technologies.

This work uses a Kintex 7 manufactured by Xilinx Inc. Despite the existence of radiation-hardened by design (RHBD) FPGAs manufactured by Xilinx, a device of
the 7-series was chosen for being a recent model, for being partially characterized for radiation environment and for the interest in using the device in critical applications [39].

2.1.1 Xilinx 7-series Architecture

The Kintex 7 is based on a SRAM memory, which does not need to be refreshed periodically. In Figure 2.1 an SRAM cells to store a bit of information with four transistors is depicted. The state adopted by the cell can be 1 or 0. When a 0 is present at the switch, the transistor creates a connection between two hard-wired segments; in the other case, the transistor exhibits a high resistance, opening the path between the segments. The bit stored in the SRAM cell may be flipped by radiation effects inside the circuitry [12], [13].

![Figure 2.1: CMOS SRAM cell.](image)

Xilinx FPGAs consist of Configurable Logic Blocks (CLBs) surrounded by Input/Output Blocks (IOBs), all interconnected by routing resources. Each CLB has a set of Look Up Tables (LUT), multiplexors and flip flops. The CLBs are tiled across an FPGA with routing wires between them, as shown in Figure 2.2.

An LUT is a logic structure capable of implementing a given Boolean function. The LUTs on the Kintex 7 are LUT 6,[40] which can implement any 6 input 1 output
boolean function, 5 input 2 output boolean function, and so on. The CLBs provide
the functional elements to build logic while the IOBs act as the interface between the
package pins and the CLBs. The CLBs are interconnected by the General Routing
Matrix (GRM) located at the intersections of routing channels. FPGAs can also have
embedded Block of RAM (BRAM), hard processors, Phase Lock Loops (PLLs), hard
Digital Signal Processing Slices, and other components that vary depending on the
FPGA series.

Figure 2.2: Tiled CLBs [53].

Figure 2.3 shows a general representation of the Xilinx FPGA architecture [15].
The characteristics of CLBs are shown. Embedded memory is also highlighted in the
general device layout.
2.1. FPGA ARCHITECTURE

2.1.2 Reconfiguration

The Kintex 7 can be reconfigured as others SRAM FPGAs. The process may be performed after powering up the device. The process consists of loading a bitstream or configuration file into the configuration memory of the FPGA. Once the bitstream is loaded and the FPGA is working, the user can keep track of the configuration memory content by performing a readback. A readback reads the content of the configuration memory and can be done through SelectMAP, ICAPE2, and JTAG interfaces[39]. It is important to check the content of the configuration memory periodically as it will be changed when exposed to radiation environments.

Figure 2.3: Xilinx FPGA Layout [15].
FPGAs can also be reconfigured without interrupting the functionality. A complete bitstream can be loaded or one can prefer to load a smaller portion. The Partial Reconfiguration (PR) allows to modify a design in an active FPGA. A small bitstream should be loaded into a frame to achieve a PR. Figure 2.4 shows the concept behind PR; where a portion of the FPGA can be used to implement several functions.

![Figure 2.4: Partial reconfiguration functionality [26].](image)

The portion labeled 'A' represents the reconfigurable logic. A1.bit, ..., A4.bit, represent the different bitstreams that can be loaded into the FPGA.

As mentioned previously, most resources are controlled by volatile memory cells.[51] The set of cells is known as the configuration memory. The configuration memory defines the behavior of:

- LUT boolean functions.
- Routing signals.
- Voltage standards for IOBs.
- Multiplexor connections.

However, configuration memory does not interact with the content of hard cores,
embedded BRAMs and data within Flip-Flops. The content and state of these resources cannot be checked by reading the configuration memory. Moreover, when doing a partial reconfiguration, the state of these elements will not be changed. If an error exists in the content of a BRAM, it will stay there even after reconfiguration. A full power on cycle will restore the content to the initial state. The existing solutions to cope with inaccessible resources is discussed in the section of mitigation techniques.

Bitstreams are divided into frames. Frames are the smallest addressable unit of the configuration memory. A frame is a 32-bit word [53]. The configuration operations should be over one or more frames. The words per frame in the 7 Series are 101. Figure 2.5 shows the basic idea behind frames for a previous family of Xilinx FPGAs.

![Figure 2.5: Frames over FPGA layout](image)

Even though, the region corresponding to several consecutive frames can be determined using tools like PlanAhead [54]. Xilinx does not provide the mapping between the bits in the bitstream and the actual resource that they control. Despite
the fact, performing a readback in the configuration memory and modifying corrupted bits is enough to cope with radiation effects on FPGAs.

2.2 Radiation Overview

2.2.1 Radiation Environment

The space environment consists of several environments that may interact with electronic devices inducing undesirable effects. The nine specific environments, considering the natural environments and the man-made environments are:

1. Neutral thermosphere;
2. Thermal;
3. Plasma;
4. Space debris;
5. Solar;
6. Ionizing Radiation;
7. Geomagnetic field;
8. Gravitational field;

The environments may affect electronic devices by generating effects on the materials. It is important to know, study and research on the effects to operate reliably in space. For this thesis, the most important effects are the ones produced by the ionizing radiation.
2.2. RADIATION OVERVIEW

The ionizing radiation environment is of particular interest in this work for it encompasses the sources of particles affecting electronic devices and causing bit flips in FPGAs memory elements. The particles of concern are electrons, protons, photons, alpha particles and heavy ions[55].

The space particles can be classified into two primary categories: [56]

- photons,
- charged particles.

The photon particles have zero rest mass and are electrically neutral. They interact with target atoms producing energetic free electrons. The charged particles interact with the silicon atoms causing excitation and ionization of atomic electrons.

The primary sources for ionizing effects are [57]:

- protons and electrons trapped in the Van Allen belts,
- heavy ions trapped in the magnetosphere,
- galactic cosmic ray protons and heavy ions,
- heavy ions and protons from solar flares.

Figure 2.6 illustrates the charged particles sources.

2.2.2 Radiation Effects on FPGAs

Radiation Characteristics

Three definitions are widely adopted to describe the radiation environment and the interaction with electronic devices:
2. BACKGROUND INFORMATION

Figure 2.6: Charged particles in space environment [56].

- Flux.
- Fluence.
- Cross section.

Figure 2.7 shows the interaction between the incident radiation in a unitary area $a$ on a surface $S$. Given a time $t$, the number of particles $p$ that crosses the unitary area will characterize the target. Each particle of the beam can have a different incident angle $\alpha$ when crossing the surface.

The particles flux is the number of particles $p$ that crossed the area $a$ on a unit of time $t$. The flux is measured in $[#particles/cm^2s]$.

The particle fluence $\Phi$ is the integral of the flux in time. The fluence describes the number of particles that crossed the section during a particular time. Fluence is
2.2. RADIATION OVERVIEW

Figure 2.7: Flux on a target area [59].

measured in $[#\text{particles}/cm^2]$.

Finally, the cross section $\sigma$ is a hypothetical measure that represents the device accurately. Such that if a particle crosses the target area, there will be an interaction. The cross section is measured in $[cm^2]$. The cross section is an important aspect for characterizing electronic devices for different particles. In this work the cross section determined by experiments was the dynamic neutron cross section, as the architectures were operating while radiated with a neutron beam.

The physical effects due to the impact of radiation on the materials used for electronic devices are phenomena too complex to model. Consider the phenomena depicted in Figure 2.8 where a primary particle strikes a material and penetrates with a scattering path. The path is random and determine by the collision of the particle with the core of the material. Even though, the path can be approximated to a linear path, the effect of the loss of charge, the secondary particles and stopping power, makes the phenomena not practical for electronic devices.
2. BACKGROUND INFORMATION

Figure 2.8: Particle actual path [59].

Simulation of the physical effect becomes extremely complex and time consuming. Fortunately, common behaviors have been identified within the affected devices. The behaviors are related to the circuit functionality instead of the physical principles of the technology, becoming more comprehensive for the user. Based on the need of a comprehensive model, the fault model is introduced [59].

Fault Model

Instead of dealing with the phenomena through mathematical models, it is possible to represent the effects of radiation considering the correct device operation. Mainly, the models define the difference between a faulty system with the behavior of a correct system.

According to NASA Fault Management Handbook [45] an abnormal state of a system includes three terms: fault, error, and failure. A fault is usually understood as the cause of an error and an error as the cause of a failure. In the standard ISO 26262, a fault is defined as an “abnormal condition that can cause an element or an item to fail.”
The error is defined as the “discrepancy between a computed, observed or measured value or condition, and the true, specified, or theoretically correct value or condition.” Finally, the failure is defined as the “termination of the ability of an element, to perform a function as required.”

Consider that in an FPGA a bit flip occurs in a register. If that bit flip does not affect the operation of the FPGA, then it is a fault. Suppose that the fault propagates, and a computation is performed with the faulty value. Then, the fault caused an error in the FPGA. If the result with errors is output or used in the FPGA, i.e. the value with errors is not masked or corrected before operating with it, the FPGA will present a failure.

Although the failure is always caused by a fault, a fault does not necessarily lead to a failure. In an FPGA circuit design, such a fault could be a flipped bit in a flip-flop or a reprogrammed logical operation due to a falsified look-up table (LUT). In any case, only if the faulty resource is actually used in the design the associated fault will finally lead to a failure.

The fault models induced by radiation can be classified into two categories:

- Single Event Effects (SEE).
- Total Ionization Dose (TID).

The difference between having an SEE or accumulating TID is based on the Linear Energy Transfer (LET) of the particle and the threshold of the material. If the LET of the particle is less than the threshold LET of the material, the particle will only accumulate. On the other hand, if the LET of the particle is greater than the LET threshold, then the particle produces an SEE. The LET is the energy transferred to the material in the path due to secondary particles emitted during the interaction [59]. The threshold refers to the energy needed in the device to present an effect in the operation of itself.
Single Event Effects

SEEs are the models of the funneling effect depicted in Figure 2.9. The SEE are induced by a single particle that hits the device in a particular spot. A different effect is produced depending where the particle struck, the moment when it hit and the energy of itself. The fact that a particle hits a device does not necessarily mean that an SEE will be produced.

![Funneling effect in CMOS transistor](image)

Figure 2.9: Funneling effect in CMOS transistor [59].

The SEEs may be transient faults which affect the device for an amount of time, until the device is reset, these are known as soft errors. If the fault is permanent it damages the device and it is known as hard error. The classification based on the effect is the following:

- Soft Errors
2.2. RADIATION OVERVIEW

- Single Event Upset (SEU).
- Multiple Cell Upset (MCU).
- Single Event Transient (SET).
- Single Event Functional Interrupt (SEFI).

• Hard Errors
  - Single Event Latch-Up (SEL).
  - Single Event Gate Rupture (SEGR).

The effects of high-energy particles are known as Single Event Effects [60]. The SEEs on electronic devices should be considered previous to utilizing an FPGA in radiation environments. Researchers have thoroughly studied the SEE on FPGAs. The most relevant effects on FPGAs are the following:

- Single-event upsets (SEU): A single-event upset is a change of a bit in a memory element. SEU are caused by ionizing radiation. The effect of radiation on memory elements may change the voltage levels on a memory cell, causing a bit flip. The modified value will be preserved on the memory element; the original value will be lost. This SEE does not cause permanent damage.

- Single-event transient (SET): A single-event transient is a temporary voltage increase due to radiation. The effects of SETs are transient glitches that propagate through an architecture. A SET becomes a SEU when a glitch is stored in flip-flops or latches. SETs do not cause permanent damage to the device.

- Single-event latchup (SEL): Single-event latchup is a destructive effect caused by charged particles which activate a parasitic silicon controlled rectifier (SCR). The impedance of the path decreases causing an increased current flow between power and ground. The current is, in many cases, large enough to cause permanent
damage to the device. Due to their destructive nature, all devices considered for space applications should be tested for SELs. Once a SEL is presented, the device’s power supply must be interrupted to stop the latchup from destroying the device.

- Single-event functional interrupt (SEFI): A single event functional interrupt (SEFI) refers to a SEE that modifies the functionality of the device. Usually, SEFIs are presented as changes in internal states of control registers. The probability of having a SEFI is small despite the fact that SEFIs can be presented in several situations, such as device reset, power-on reset, initiation of operating mode, lockup, sleep mode, to name a few. The main reason for a low probability is that the area devoted to these structures is very small. SEFIs do not cause permanent damage to the device.

This thesis considers soft errors for the estimation of the reliability as they are more prominent in the space environment. No distinction is made of the type of soft error causing the failure in the architecture. The data collected in neutron testing experiments is the amount of fluence needed to cause a failure in the architecture. Moreover, SEL have not been found in the device under test [35]. As for SEGR, if an FPGA is damaged during testing two more boards can be used to keep the experiments going. Section 2.3 discusses the manifestation of SEE in FPGA and how to cope with them.

**Total Ionizing Dose**

The amount of radiation dose that a device can tolerate before failing to meet published parameter specifications is called total ionizing dose (TID). [63] The TID is the effect of accumulating charge. If the exposition time increases or the flux increases, the device accumulates more charge.
2.2. RADIATION OVERVIEW

The TID in space applications and avionics is mainly caused by the effect of protons and electrons present in the Van Allen belt, depicted on Figure 2.10; and by the constant interaction of secondary particles with the device itself. The TID accumulation produces several effects like wearing out of the device, slow response of transistors, more power consumption, increases on the sensibility to other SEEs.

![Figure 2.10: Charged particles in Van Allen belts [56].](image)

TID is formally the accumulated energy transferred to a material per mass unit due to the interaction with particles. TID is measured in [rad], and acronym for radiation absorbed dose. 1 [rad] is equal to $0.1 \frac{J}{kg}$. 

---

<table>
<thead>
<tr>
<th>page</th>
<th>content</th>
</tr>
</thead>
<tbody>
<tr>
<td>25</td>
<td>TID is formally the accumulated energy transferred to a material per mass unit due to the interaction with particles. TID is measured in [rad], and acronym for radiation absorbed dose. 1 [rad] is equal to $0.1 \frac{J}{kg}$.</td>
</tr>
</tbody>
</table>
2.3 FPGA Fault Mitigation Techniques

2.3.1 Reliability Overview

Fault Mitigation Techniques are implemented to increase the reliability of the system. The reliability $R(t)$ is the probability that a device operates correctly under certain conditions during the interval of time $(t_0, t)$, considering that it was working properly at time $t_0$. 'Working correctly' means that no failure –end of the ability to perform a required task– should be presented [51].

Consider a test on $N$ components, statistically equals, at time $t_0$ and registering the number of components that failed and the number of components that worked correctly at time $t$. The number of components that failed in time $t$ are $N_f(t)$ and the number of components that remain working correctly are $N_o(t)$. The reliability of the component in time $t$ is shown in equation 2.1; which is the probability that a component survives to the interval $[t_0, t]$. The expression assumes that if a component fails it will stay that way.

$$ R(t) = \frac{N_o(t)}{N} = \frac{N_o(t)}{N_o(t) + N_f(t)} \tag{2.1} $$

The probability that a component does not survive in the interval $[t_0, t]$ is called unreliability $Q(t)$. The expression of the unreliability is shown in equation 2.2.

$$ Q(t) = \frac{N_f(t)}{N} = \frac{N_f(t)}{N_o(t) + N_f(t)} \tag{2.2} $$

For any given time $t$, $R(t) = 1.0 - Q(t)$.

The reliability can be written as a differentiation of $R(t)$ with time, as shown in equation 2.3. The derivate of $N_f$ with time represents the failure rate of a component.
In time $t$ there are $N_o$ components working correctly. Dividing the instantaneous failure rate by $N_o(t)$, results in equation 2.4 which represents the failure rate. The failure rate is measured in faults per time unit.

\[
\frac{dN_f(t)}{dt} = (-N) \frac{dR(t)}{dt} \quad (2.3)
\]

\[
z(t) = \frac{1}{N_o(t)} \frac{dN_f(t)}{dt} \quad (2.4)
\]

If the assumption that the failure rate is constant is made and has a value of $\lambda$, the solution to the differential equation is shown in equation 2.5.

\[
R(t) = e^{-\lambda t} \quad (2.5)
\]

Where $\lambda$ is the constant failure rate.

The exponential relation between the reliability and time is known as exponential law of failures [20]; which states that for a constant failure rate the reliability changes exponentially with time.

The law is successfully used in the analysis of electric components. Moreover, the exponential model is related with Poisson, which allows expressing the probability $P(t, n)$ that in an interval $(0, t)$ $i$ events will occur.

### 2.3.2 Fault Mitigation

Several FMTs for FPGAs have been performed [5], [36], [28], [38]. The main goal is to increase the reliability by tolerating failures. Failure tolerance can be achieved either by failure masking or by failure recovery[5].
Redundancy usually produces failure masking. Spatial redundancy is very common, techniques such as: Duplication With Compare (DWC) [29], Triple Modular Redundancy (TMR) [41] and Reduced Precision Redundancy (RPR) [30] are fairly recognized techniques. Information redundancy consists of Error Detection and Correction (EDAC) algorithms and Algorithm-Based Fault Tolerance (ABFT).

In addition to failure masking techniques, failure recovery techniques can be used to increase the reliability. The most common technique is scrubbing [36], [5], which consists on refreshing the configuration memory to correct bit flips caused by SEUs. Fault recovery techniques can also be applied to BRAMs or other memories by following the same idea of scrubbing [31].

Several scrubbing techniques have been proposed in the literature. The scrubbers fall in at least one of the following categories:

- Blind and Readback scrubbing.
- Device and Frame-Oriented scrubbing.
- Periodic and On-Demand scrubbing.
- Internal and External scrubbing.

Within an FPGA all the elements have a different vulnerability to SEEs. The configuration memory and the BRAMs are the most vulnerable [64], [18]. In Table 2.1, techniques to cope with SEEs in different elements of an FPGA are presented.

### 2.4 FPGA’s Architecture Testing

In order to identify the sensitive modules within an FPGA-based system, several tests may be conducted: life testing, accelerated radiation test, fault injection and analytical
2.4. FPGA’S ARCHITECTURE TESTING

<table>
<thead>
<tr>
<th>FPGA element</th>
<th>Problem caused by SEE</th>
<th>Possible mitigation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configuration Memory</td>
<td>Corrupts circuit’s operation, bus conflicts, modification of logic</td>
<td>Partial or full reconfiguration</td>
</tr>
<tr>
<td>CLBs</td>
<td>Transient error propagation in the logic</td>
<td>System level redundancy</td>
</tr>
<tr>
<td>BRAMs</td>
<td>Corruption of stored data</td>
<td>TMR, Error Detection and y Correction Codes (EDAC)</td>
</tr>
<tr>
<td>IOBs</td>
<td>False outputs to other devices or false inputs</td>
<td>TMR</td>
</tr>
</tbody>
</table>

Life testing consists of placing the system under the natural radioactive environment. The drawbacks are the reduced amount of data that may be acquired, the long duration of the test, the high cost of placing the system under the needed environment and the enormous quantity of devices needed to provide an acceptable sample.

In the accelerated radiation testing, the system is exposed to a particle flux several orders of magnitude greater than in the space [46]. So, statistics equaling years of radiation exposure can be collected in just a few minutes. Moreover, with this test, different parameters (Weibull, Edmond) can be found for complete characterization of the system in matter. The drawbacks of this test are the expensive facilities and the need to perform this test with each new application synthesized on an FPGA device.

Radiation testing can also be performed with neutron beams. Neutron beams throw a lower flux that will not gather statistics as great as with another beam. However, they are less expensive and have proven useful to estimate the dynamic cross section and to verify the functionality of an architecture to upsets. The effect caused by neutrons is physically different from the effect of charged particles, but the manifestation of the effect is the same: a bit flip.
The fault injection test can be done as an additional feature for debugging. The basic concept of this technique is to insert a fault and observe how it propagates through the system. The drawbacks are for large scale systems where inserting faults all over the systems results in a prohibitive time for the experiment to perform correctly. It is also noteworthy that no matter what injection tool is used, the results will always depend on the workload [65].

Lastly, analytical techniques aim at studying SEUs and Multiple-Bit Upsets (MBU) in the configuration memory of the device. The technique identifies all possible sources of errors. The drawbacks are that tools like STAR [14] do not consider MBUs in Input/Output Blocks (IOB) and in Blocks of RAM (BRAM). Likewise, the results of STAR provide all the possible sources of errors, but they are not related to the space environment.
This chapter reviews previous works related to the reliability estimation of FPGAs for critical applications. The chapter starts by briefly reviewing the widely used approach of modeling with Markov chains. The next section discusses the challenges of estimating the reliability for FPGA architectures. Followed by a review of the tests performed to compute the reliability. Lastly, the importance of placing majority voters on triplicated design is discussed.

3.1 Markov Modeling

Reliability can be estimated with combinatorial models [21]. Combinatorial models use probabilistic techniques that enumerate the different ways in which a system can remain operational. The two models that are most common are series and parallel.
3. RELATED WORK

The primary difficulty with the combinational models is that many complex systems cannot be easily model in a combinatorial fashion. Moreover, the fault coverage is not easily incorporated into the combinatorial models. For these reasons, Markov chains are commonly used to estimate the reliability[20].

A fundamental problem in fault-tolerant computing is predicting the system’s reliability. Markov modeling is the preferred method to estimate the reliability of FPGAs as Markov chains are flexible enough to represent the behavior of architectures synthesized on FPGAs [22]. The underlying assumption of Markov models is that the probability of a state transition depends only on the current state. These are called first-order Markov models [23], [21], [20]. The architectures analyzed in this thesis can all be modeled with first-order Markov models.

In Markov models, each state represents a combination of faulty and fault-free modules. The transitions are probabilities such as the probability of failure or probability of recovery.

On [22] authors explore the use of Markov models to estimate the reliability of Triple Modular Redundancy (TMR) systems. Assuming a failure rate and recovery rate, the authors estimate the reliability for different TMR versions. The TMR versions explored are the system TMR, partitioned TMR and non-persistence error. A persistence error is a permanent interruption experienced in an FPGA architecture with scrubbing [24], [25]; while a temporary interruption is called a non-persistence error. The authors conclude that TMR along with scrubbing highly increases the reliability of a system. Moreover, authors show a plot of partitioning TMR with two and three partitions, concluding that more partitions lead to an improve of the reliability. Nevertheless, partitioning an architecture may be challenging as voters should be placed on all inputs and outputs of each portion.

For complex FPGA designs estimating the reliability with a Markov model may become challenging. It becomes even more challenging after applying FMTs [20].
Nevertheless, dealing with the hurdle leads to a low-cost and fast method to estimate the reliability performance of the architecture.

3.2 Reliability Challenges

Nowadays, estimating the reliability of FPGA architectures is a challenging task. Devising a mathematical model out of an FPGA architecture is not a direct process. The complexity of today’s architectures makes the use of radiation testing a more suitable option than estimating the reliability by mathematical means [58].

Currently, many researchers validate their design by radiating with a neutron beam rather than with a proton or heavy ion beam [32], [33] for being cheaper and easier to track the bit-flips along the architecture. By using neutron testing the increase in the reliability or decrease in the dynamic cross section after using FMTs and the proper operation of the FPGA can be evaluated.

The work on [32] uses neutron testing to evaluate the dynamic cross section of a system based on a processing core to compute the Fast Fourier Transform with four different granularity levels of TMR. The goal is to show the impact of the grain used in TMR and the impact of doing full against partial reconfiguration on the cross section of the system. The four TMRs synthesized differ in the number of voters and their distribution within the architecture. The remarks of the work are that partial reconfiguration improves the correction time; however the impact on the failure rate is negligible.

While many FMTs can be found in the literature, the majority do not report the reliability achieved by the proposed technique. There are few works which indicate the reliability of a system when applying a certain FMT [58], [33], [28].

In [58] the authors provide a methodology to do static and dynamic testing on
FPGAs. The static testing is a characterization of the device’s elements. The dynamic testing is proper to the architecture running during the test. The authors compare radiation testing results against fault injection and modeling tools. The authors performed radiation testing on an adder tree synthesized on a Virtex-II FPGA to validate the data of the other test methods. Fault injection is preferred for low-cost and test coverage. The model tools are based on reading EDIF files created during the synthesis of the architecture. EDIF files do not contain information about routing and placement, having no way to assess placement issues. The authors finally recommend model tools only for being faster to estimate the reliability, but their accuracy is a controversial matter.

In [33] authors present a reliability estimation of an architecture composed of 190 cascaded 16-bit adders and registers by performing a neutron test. The work proposes adaptive voters that mask the erroneous modules and selects the correct output of the remaining. The architectures are tested in an n-Modular Redundancy (nMR) where n goes from 3 to 7. The architecture 7MR was fault injected to determine the needed fault to cause a failure. All of the architectures were tested under a neutron beam. The results show an increase in reliability of 19 times of TMR against 7MR with an increase of power of only 33%.

Another work that estimates the reliability through radiation testing is presented in [28]. The authors apply TMR with scrubbing to an FPGA and report the increase of reliability. The architecture described on the FPGA was tested using a proton beam. It is understandable that making that kind of test for all FTTs available in the literature is impractical, as the facilities needed are very expensive. However, a reliability model can be obtained for estimating the specific probability of failure for an FMT within an FPGA.

In [28], the goal is to estimate the reliability with a reliability model. The proposed model considers the orbit-specific upset rates, the probability of failure given a bit upset and the usage of TMR with scrubbing. The model computes the probability of failure
given an upset in two ways. The first one is by performing fault injection on the device, waiting for the fault to propagate and identifying if a failure was presented. The second approach is by radiation testing. Both approaches can estimate the device cross section which is needed for further steps of the modeling. The radiation test results also serve to compare the fault injection results. The primary parameter in the model is the upset rate ($\lambda$) which is computed by using CREME96\textsuperscript{1} model [62] to simulate the orbital parameters and the interaction of the device with particles in the specified orbit. Authors conclude that the estimation through fault injection approach is not accurate as several point of failures were found in the architecture.

Another approach to estimate the failure rate given specified orbit parameters is by computing the cross section from reported characterization of the device. The cross section is obtained by characterizing the desire FPGAs. The preferred method is proton and ion testing. Efforts have been made for characterizing RHBD FPGAs and non-RHBD FPGAs. Currently, data regarding the Virtex II PRO, Virtex 4, Virtex 5 and the most recent Kintex 7 [44],[35] can be found in the literature. With the data reported it is possible to make pessimistic approximations of the reliability. The approach follows the use of tools based on the CREME96 model. The user specifies the characterization parameters of an FPGA and the orbital parameters. The tool will compute the radiation on the orbit that will interact with the satellite. After specifying all the data, the results will be a failure rate for the given architecture. The failure rates computed are usually lower than the failure rate measured in space missions [42].

The radiation test to characterize an FPGA are made with protons and a wide number of heavy ions. The particles have a different energy. The tests consider the static and dynamic parameters. The difference is that static is intrinsic to the device, whereas dynamic parameters are intrinsic to the architecture. A Weibull curve approximates the characterization results. The axes are the cross section obtained for different effective LETs. SPENVIS is another tool that can help in computing the failure rate given

\footnote{https://creme.isde.vanderbilt.edu/}
3. RELATED WORK

Weibull parameters and orbit parameters. SPENVIS runs the CREME96 model and other models to calculate the particles in the trajectory of the space mission [62].

In addition to characterization data, data of flight experiences are reported in a more limited way [42], [43]. However, these data are not enough for making a decision when designing an FPGA architecture for space applications. The work [42] compares the data obtained from fault injection and radiation testing. The conclusion is that the upsets suffered on the Virtex 4 architecture during 14 months of operation at Low Earth Orbit (LEO) are significantly lower than the predicted number. The predicted failure rate is between 4 to 5 times higher than the actual failure rate.

Another work reporting flight experience is presented in [43]. The work shows data collected from the Cibola flight experiment satellite. The experiment has nine FPGAs used in the payload for high-throughput sensor processing and monitoring SEUs. There is not a contrast between previously estimated failure rates and the actual rates. The work concludes that the FPGAs are suitable for high-processing task on space. Regarding SEUs, the authors concluded that the FPGAs at Cibola experiment has performed very well. Finally, the work states that with the ongoing improvements of FPGAs, they will have an advantage in terms size reduction and power.

Efforts to estimate the reliability with mathematical models have been scarce. In [34] the authors proposed a method to estimate the reliability by using GPSS software. GPSS software is used to generate data regarding SEUs. The architecture tested is a TMR with an external scrubber. Likewise, the voters are assumed to have a reliability of '1'. The program simulates transactions that appear at a given rate (failure rate). Those transactions, simulated faults, are stored into one of the three modules conforming the TMR. If a module fails, the other two act as a DWC and the faulty module is reconfigured. However, if two modules fail all the modules are reconfigured. In the work, first, a mathematical model is done for the mitigated architecture. Then, the model is validated with the program made on GPSS. The results show that the plots
from the mathematical model and the GPSS program are fairly similar, with the GPSS results being more pessimistic.

Another effort was made for microcontrollers. [16]. The authors report an approach to studying the effects of upsets on microcontrollers. The idea is to inject bit flips randomly and simulate faults where bit injections cannot be made. The approach needs extra hardware to simulate upsets on interruptions, to monitor expected results and to monitor the execution time. As way of conclusions, the approach of Code Emulated Upsets (CEU) yields similar results when compared to radiation testing, confirming the potential offered by the CEU approach to predict error rates of real applications. This approach will be interesting to take on FPGA architectures; unfortunately, the approach has not been taken for FPGAs yet.

Even with the guidelines provided by NASA Goddard Space Flight Center choosing a Fault Mitigation scheme for a given application is challenging [49], [50]. The guidelines specify methodologies for developing FPGA designs. They are intended for the use of designers that will use FPGAs. They are concerned with the development process and with the detailed design guidelines. The biggest hurdle is to estimate the reliability by testing the design. Several testing methods used nowadays are described in Table 3.1 along with their drawbacks.

The fact that all test and methods to estimate the reliability have wide drawbacks is clear. One particular drawback of the testing methods is that they provide pessimistic estimations of reliability. When actually placing the FPGA in the radiation environment the actual failure rate is lower than the one predicted. Even though it is alway safe to assume a factor of error so that the design tolerates more radiation that the one estimated, the predicted failure rate can be more than 5 times greater than the actual rate. The implications of this pessimistic prediction are additional hardware, additional power consumption, and additional expenses.
### Table 3.1: Testing Methods.

<table>
<thead>
<tr>
<th>Tests</th>
<th>Description</th>
<th>Drawbacks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Life testing</td>
<td>Placing the system under natural radioactive environment.</td>
<td>Poor amount of data.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>High cost.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Long duration of test.</td>
</tr>
<tr>
<td>Accelerated radiation test</td>
<td>Exposing the system to a particle flux several orders of magnitude greater than in the space [44].</td>
<td>Expensive facilities needed.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Unavailability to perform the test iteratively.</td>
</tr>
<tr>
<td>Fault injection test</td>
<td>Insert a fault and observe how it propagates through the system.</td>
<td>Unable to inject faults to unaddressable FPGA’s components.</td>
</tr>
<tr>
<td>Analytical techniques</td>
<td>Aim at studying SEUs in the configuration memory of the device. The technique identifies all possible sources of errors.</td>
<td>Do not consider upsets in all FPGA’s components.</td>
</tr>
</tbody>
</table>
3.3 Voters Placement

The authors in [32] report another interesting finding. The effect of the distribution of majority voters throughout the architecture has a significant impact on the reliability. Four architectures were tested. Those architectures are:

- A CGTMR with voters at the output of the architecture.
- A FGTMR where the FMT architecture is broken into 12 TMR logic blocks with voted inputs and outputs.
- A FGTMR based on architecture number 2 but deleting the voters of five blocks.
- A FGTMR based on architecture number 2 without voters on seven blocks.

Authors conclude on the importance of the placement of majority voters. The most reliable architecture was the one broken into 12 blocks, surprisingly followed by the architecture broken into five blocks. The architecture with only five blocks is stated to be more reliable than the architecture with seven blocks. Even with fewer majority voters an architecture can have a smaller cross section if the voters are placed on critical parts of the design.

However, more evidence is needed to establish a method to place the least number of voters to achieve the highest reliability. The placement of the voters in the FMT architecture was made based on its functional blocks. The decision of eliminating voters in the blocks was made considering the importance of the arithmetical blocks. Architecture 3 removes the voters surrounding the arithmetic blocks, while architecture 4 only keeps the voters surrounding the arithmetic blocks.

Automatic tools exist to place voters along an architecture. The most recognized tool is XTMR [47] proposed by Xilinx. The tool triplicates all the logic and majority voters are inserted in the feedback of the flip-flops to repair bit flips. The output of the
Another tool is the BYU-LANL TMR (BLTMR) tool which identifies structures corresponding to persistent configuration bits. Primitives that are part of feedback structures contribute to persistent error behavior. The BLTMR first triplicates the structures on feedback loops and add majority voters. If resources are not used entirely, the BLTMR will triplicate additional logic [48]. Architectures with BLTMR have been tested under radiation beam achieving an improved reliability close to 100 times.

3.4 Remarks

In this thesis, a method to test FPGA architectures is presented. The method needs the reliability of the unmitigated FPGA architecture. Afterward, mitigation techniques can be applied. The method dictates a guideline to follow to estimate the reliability of the FPGA architecture with mitigation techniques. Furthermore, it will help to decide which FM scheme to apply, as the use of the proposed method does not present a substantial cost, and several FM schemes can be tested. Even though most of the research discussed in the thesis is related to radiation environments, the test method will serve in other industrial areas where reliability is a concern.

The work presented in this thesis also provides evidence on the importance of the majority voters placement. Placing an increased number of voters cannot be automatically translated into an increased reliability. The distribution of the voters has a due importance in the task of reducing the cross section in an FPGA.
Markov chains have been widely used to estimate the reliability of systems. The ability to integrate features such as fault coverage and module recovery make them suitable for reliability modeling of FPGA architectures. Moreover, using a mathematical method is lower in cost than iteratively testing the architecture to estimate the reliability.

In this thesis, Markov chains are utilized. As usual, the inputs needed for the proposed models are the recovery rate and the failure rate. The recovery rate is the rate of an external scrubber. The reliability of this scrubber is 1; meaning that the scrubber will not fail. The assumption is based on the fact that the scrubber is external and is not under the beam during the test. For space missions, a scrubber with reliability close to 1 to SEUs can be achieved by using a rad-hard component or the main onboard computer.

The failure rate used on the models comes from experimental data of the
unmitigated design. The data thrown by the experiment can be used to compute the neutron cross section and the neutron failure rate of the architecture. The latter is the input to the proposed models. The experimental setup and the unmitigated design are thoroughly described in and Chapter 5.

The output of the model is a transition matrix which can be placed in a set of differential equations. The solution to the set gives the probability of being in each state. The addition of the probabilities of all the functional states is the reliability. The addition of the failure states is the unreliability. The reliability can be computed in terms of the unreliability as follows:

\[ R(t) = 1 - Q(t); \] (4.1)

where \( R(t) \) is the reliability at any time \( t \) and \( Q(t) \) the unreliability.

The model can estimate the reliability of a mitigated design with two inputs: the failure rate of the unmitigated design and the area of each portion of the mitigated design. Each portion corresponds to the logic comprised between voting elements. The model does not need characterization parameters of the FPGA under test, making the model flexible to compute the reliability on a new FPGA and on FPGAs where no characterization data exists.

4.1 Proposed Method

In the proposed method of this thesis, a Markov chain is used to estimate the reliability of an architecture with Fault Mitigation Techniques (FMT). The inputs are the failure rate of the unmitigated design and the scrubber rate. As output, the model gives the probability of being in each state of the model, which can be manipulated to have the reliability of the architecture.
4.1. PROPOSED METHOD

The steps involved in the estimation of the reliability are shown in Figure 4.1.

![Figure 4.1: Block diagram of the main steps to estimate the reliability.](image)

The description of the steps is the following:

- **Synthesize unmitigated architecture.** The data needed for the model is the failure rate of this architecture. A wrapper with high reliability should be used to count the failures presented on the ongoing radiation test.

- **Add FMTs to the architecture.** FMTs improve the reliability of the architecture. TMR is the most used FMT, along with a scrubber for preventing faults to accumulate. Other FMTs can be employed too.
  - Identify the level of redundancy, i.e. number of modules.
  - Recognize the type of scrubber to employ.

- **Analyze resources between voters.** Voters are placed when TMR is applied. TMR can be Coarse Grain (CG), Medium Grain (MG) or Fine Grain (FG) depending on the number of voters added and where they are added. This work applies MG to have a reduced number of voters allowing to estimate the resources between...
4. RELIABILITY MODEL

the voters.

– Label the portions of the architecture between voters. The logic between voters is input to the proposed model; to estimate the resources the steps listed below are followed:

  * Spot isolated resources. Isolated resources are a group of resources with all their inputs and outputs voted.

  * Count the number of portions. The addition of all the resources in the portions should account for exactly the resources utilized by architecture to ensure the least number of portions.

  * Estimate the resources for each portion.

  * Estimate a percentage of the total area of each portion.

  * Obtain the failure rate of each portion by multiplying the percentage with the total failure rate.

– Identify shared resources for each pair of portions.

– Compute the percentage of the total area.

• Develop a Markov Chain of the mitigated architecture. At this point, all inputs of the model are known.

  – Identify the number of states needed to represent the behavior of the architecture. The states represent a distinct combination of faulty and fault-free modules.

  – Add the transitions, given by the failure rate and repair rate.

• Obtain a system of equations where the equations are the probability of being on each state. Equations can be easily taken from the transition matrix of the
4.2 PROPOSED MODELS

Markov chain.

• Solve the system of equations. Each equation expresses the probability of being in a certain state. To compute the reliability, all the probabilities of functional states should be added up. To compute the unreliability all the failure states should be added up.

• If the unreliability was computed, subtracting the unreliability of the architecture from '1' to get the reliability.

• Plot the reliability.

4.2 Proposed Models

Two models were developed following the proposed method, to highlight the importance of shared resources in a Markov model. The first model assumes equal portions for the three areas identified between the four voters added and no resources shared between the portions. The assumptions were made based on analyzing the layout of the architecture. The second model has three uneven portions dividing the architecture’s resources. Likewise, the second model has a portion representing the sharing resources between portions of the architecture.

The models represent a MGTMR using Markov chains. Both models have an external scrubber. The detailed description of the first model is presented below.

4.2.1 First Model

In the first approach, the first model has five states, numbered from 0 to 4. Figure 4.2 was used to obtain the number of states and the transition values. The dashed line
indicates the division made by voters. The Figure 4.2 shows the number of modules and the three portions of the model.

![Figure 4.2: Modules and portions for first approach.](image)

Figure 4.3 shows the Markov chain of the first approach, where the three portions have the same area. $\lambda_{van}$ represents the failure rate of the vanilla design; $\mu$ is the external scrubbing rate; and $\Delta t$ is the increment of time. The states of the model are described as follows:

![Figure 4.3: Markov chain for the first approach.](image)

0. No Faults.

1. 1 fault in any module/portion.
2. 2 faults on different portions.

3. 1 fault in each portion.

4. 2 faults in the same portion.

State 4 represents the failure state. The probability of being in that state is the unreliability of the system. Once the system enters into the failure state, it cannot be recovered. Notice, that on states 1 through 3 there is a probability of completely recovering, thus returning to state 0.

To compute the probability of each transition a set of equations should be solved. To get the system of equations, it is useful to first obtain the transition matrix from the Markov chain. The transition matrix for the Markov chain is:

\[
\begin{bmatrix}
1 - 3\lambda \Delta t & 3\lambda \Delta t & 0 & 0 & 0 \\
\mu \Delta t & 1 - (\frac{2}{3}\lambda + \mu) \Delta t & 2\lambda \Delta t & 0 & \frac{3}{2}\lambda \Delta t \\
\mu \Delta t & 0 & 1 - (\frac{2}{3}\lambda + \mu) \Delta t & \lambda \Delta t & \frac{3}{2}\lambda \Delta t \\
\mu \Delta t & 0 & 0 & 1 - (2\lambda + \mu) \Delta t & \frac{3}{2}\lambda \Delta t \\
0 & 0 & 0 & 0 & 1
\end{bmatrix}
\]

Where the position represented by row \( m \) and column \( n \) is the probability of transition from state \( m \) to \( n \). Notice that for inexistent transitions a 0 is written and when the system fails a 1 is written, meaning that the system will stay in that state indefinitely.

A set of equations representing the probability of being in each state at time \((t + \Delta t)\) can be defined with the transition matrix. The set of equations for the Markov chain in Figure 4.3 is the following:
4. RELIABILITY MODEL

\[ P_0(t + \Delta t) = P_0(t)(1 - 3\lambda_{\text{van}}\Delta t) + (P_1(t) + P_2(t) + P_3(t))(\mu\Delta t) \]
\[ P_1(t + \Delta t) = P_0(t)(3\lambda_{\text{van}}\Delta t) + P_1(t)(1 - (8/3\lambda_{\text{van}} + \mu) \times \Delta t) \]
\[ P_2(t + \Delta t) = P_1(t)(2\lambda_{\text{van}}\Delta t) + P_2(t)(1 - (7/3\lambda_{\text{van}} + \mu) \times \Delta t) \]
\[ P_3(t + \Delta t) = P_2(t)(\lambda_{\text{van}}\Delta t) + P_3(t)(1 - (2\lambda_{\text{van}} + \mu) \times \Delta t) \]
\[ P_4(t + \Delta t) = P_1(t)(2/3\lambda_{\text{van}}\Delta t) + P_2(t)(4/3\lambda_{\text{van}}\Delta t) + P_3(t)(2\lambda_{\text{van}}\Delta t) + P_4(t); \]  \hspace{1cm} (4.2)

The unreliability is the probability of being in state 4, subtracting the unreliability from 1 yields the reliability. The plot on Figure 4.4 shows the reliability of the system in hours for a failure rate of 0.001 hour\(^{-1}\) and a recovery rate of 0.0417 hour.

Figure 4.4: Model 1 reliability curve.

4.2.2 Second Model

The second model is an approximation with more details. This model was made in a general fashion – with not all the assumptions of the previous model. The detailed
4.2. PROPOSED MODELS

The approach does not assume equal areas’ sizes of the portions and considers shared logic between two portions. Only one sharing-logic area was considered out of the three possible sharing-logic areas because the third portions, the BRAM portion, is practically completely isolated from other resources. Figure 4.5 shows a block diagram of the portions considered in the model.

![Diagram of portion division](image)

**Figure 4.5: Second model portion division.**

The model considers as input the repair rate $\mu$, and the failure rates $\lambda_i$ of each portion. The Markov chain for the second model is presented in Figure 4.7. The chain does not depict the transitions from each state to themselves for clarity of the representation. However, those transitions do exist, and their value is computed by subtracting all the transitions from the state in matter from 1. The states for the Markov chain of Figure 4.7 are described as follows:

0. Initial state.

1. A fault in a module of portion 1.
2. A fault in a module of portion 2.

3. A fault in a module of portion 3.

4. A fault in two modules due to an upset in the sharing resources between portion 1 and portion 2.

5. Failure state.

6. A fault in a module of portion 1 and a module of portion 2.

7. A fault in a module of portion 1 and a module of portion 3.

8. A fault in a module of portion 2 and a module of portion 3.

9. A fault in a module of each portion.

The fault of the portions for each state can be seen as the combination of the three portions plus the additional failure state. The states are summarized in Figure 4.6.

<table>
<thead>
<tr>
<th>State</th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td>FAILURE</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Figure 4.6: States’ description for the second model.

The state 5 is the failure state, once the system fails it stays there indefinitely. The state 6 and 4 are equal states, thus, 6 is treated as 4. Notice that from state 1 to 9, except for state 5, there are transitions to state 0, the fault-free state, as a successful recovery means that all faults are corrected.
4.2. PROPOSED MODELS

To compute the reliability first, the probability of being at each state should be calculated. The transition matrix obtained from Figure 4.7 is the following:

Figure 4.7: Markov chain of the MGTMR architecture.
<table>
<thead>
<tr>
<th>$\mu \Delta t$</th>
<th>1 - $3(\lambda_1 + \lambda_2 + \lambda_3 + \lambda_{c12})\Delta t$</th>
<th>$3\lambda_1 \Delta t$</th>
<th>$3\lambda_2 \Delta t$</th>
<th>$3\lambda_3 \Delta t$</th>
<th>$3\lambda_{c12} \Delta t$</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mu \Delta t$</td>
<td>1 - $(2\lambda_1 + 2\lambda_2 + 2\lambda_3 + 2\lambda_{c12} + \mu)\Delta t$</td>
<td>0</td>
<td>0</td>
<td>$(3\lambda_1 + \lambda_{c12})\Delta t$</td>
<td>$(2\lambda_1 + 2\lambda_2 + 2\lambda_3 + 2\lambda_{c12} + \mu)\Delta t$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$\mu \Delta t$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1 - $(3\lambda_1 + 3\lambda_2 + 3\lambda_3 + 3\lambda_{c12} + \mu)\Delta t$</td>
<td>$(2\lambda_1 + 2\lambda_2 + 2\lambda_3 + 2\lambda_{c12} + \mu)\Delta t$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$\mu \Delta t$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>$(2\lambda_1 + 2\lambda_3 + 2\lambda_{c12} + 2\lambda_1 + 2\lambda_2 + 2\lambda_3 + 2\lambda_{c12} + \mu)\Delta t$</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$\mu \Delta t$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>$(2\lambda_2 + 2\lambda_3 + 2\lambda_{c12} + 2\lambda_1 + 2\lambda_2 + 2\lambda_3 + 2\lambda_{c12} + \mu)\Delta t$</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$\mu \Delta t$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>$(2\lambda_1 + 2\lambda_2 + 2\lambda_{c12} + 2\lambda_1 + 2\lambda_2 + 2\lambda_3 + 2\lambda_{c12} + \mu)\Delta t$</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
From the matrix, the probability of being at state 5, the failure state, at time \((t + \Delta t)\) is:

\[
P_5(t + \Delta t) = 2P_1(t)(\lambda_1 + \lambda_{c12})\Delta t + 2P_2(t)(\lambda_2 + \lambda_{c12})\Delta t + 2P_3(t)\lambda_3\Delta t + 2P_4(t)(\lambda_1 + \lambda_2 + \lambda_{c12})\Delta t + 2P_5(t) + 2P_7(t)(\lambda_1 + \lambda_3 + \lambda_{c12})\Delta t + 2P_8(t)(\lambda_2 + \lambda_3 + \lambda_{c12})\Delta t + P_9(t)(2(\lambda_1 + \lambda_2 + \lambda_3) + 3\lambda_{c12})\Delta t;
\]

Equation 4.3 represents the unreliability. The reliability is computed by subtracting the unreliability from 1. Figure 4.4 depicts the reliability of the system in hours for a failure rate of \(0.001 \text{ fault hour}^{-1}\) and a recovery rate of \(0.0417 \text{ hour}^{-1}\).

Both models take as input the results of the radiation testing on the unmitigated design; and the recovery rate from the external scrubber. Namely, \(\lambda = 0.001 \text{ fault hour}^{-1}\) and the recovery rate in hours of \(\mu = 0.0417\). Detailed information of the tested architectures can be found in Chapter 5.

It is worthy to point out that both models have exactly one failure state, making easier the computation of the unreliability and then computing the reliability. From
model 1 to model 2 the increase of states was only of four states. The states on the first approach increase linearly as more portions are identified. Having $p$ portions the number of states of the first approach will be $p + 2$.

The models are also close to each other reliability-wise. Both plots are similar to each other. The first approach shows a more reliable architecture than the one depicted by the second approach. The addition of sharing resources for the second model increases the probability of failure of the architecture, having a lower reliability than the first approach.
ARCHITECTURE UNDER TEST

Three architectures based on the PicoBlaze (PB) microprocessor were synthesized for further neutron testing. The architectures are:

- Vanilla.- The baseline design with a scrubber in the BRAM.
- Medium Grain TMR.- The triplicated design with 4 triplicated voters.
- BYULANL TMR.- The triplicated design with 43 additional voters.

The PB softprocessor was chosen as it is a well-suited solution to substitute complex state machines. PB allows controlling several resources in a small and compact fashion.

Moreover, a PB architecture was chosen as an additional branch of research. Even though Xilinx [52] publishes PB reliability for neutron effects on Earth, few is the data published from radiation experiments regarding PB reliability on the 7-series. Another
reason is the fact that the model needs to count the resources between voters of two architectures. PB is a softcore where Register Transfer Logic (RTL) diagram can be explored through Vivado. While it is true that a PB architecture is challenging to keep track of all signals, the PB architecture represents a rewarding case of study as it shows that the model works with complex architectures.

Figure 5.1 shows the block diagram of the PB, the areas of interest to monitor are highlighted in blue. The areas were hand-picked as they provide an overview of the sensitive elements in the PB. The signals monitored are 12-bit address, 8-bit output and 18-bit instruction.

The three architectures run the same program which has roughly 500 instructions, uses 16 registers, has interruptions disabled and stacks the PC 16 times. The total utilization of the scratchpad RAM was $\frac{1}{4}$ of the total capacity. The program runs seven routines:

1. Counter in registers and scratchpad. Stack PC Counter.
2. NOP performed with jump and copy on the same register.
3. Printing from registers and scratchpad.
4. Comparison and cleaning the registers.

5. Load values on register and scratchpad for printing.


7. ALU operations.

5.1 Vanilla Design

The first architecture based on PB is the vanilla design (VD). The VD only has a scrubber for the BRAM in the PB. The BRAM is where the instructions are stored for the PB, it is seen as the Program Memory (PROM) of the soft processor. A block diagram of the BRAM scrubber is shown in Figure 5.2.

The scrubber is a technique that refreshes the memory content when a fault is detected. The faults are detected using Triple Modular Redundancy (TMR). When TMR detects a fault, the faulty data in the memory location is refreshed with the data of a majority voter.

An additional scrubber was used during the neutron radiation of both architectures. The external scrubber was not under radiation. The scrubber completes a scrubbing cycle in 2.54 seconds via JTAG. The scrubber read the configuration memory and compares it with a golden bitstream. When the scrubber finds a faulty bit, it flips the bit back again to the correct value.

The PB architectures were replicated within the design due to their low resources utilization. The idea was to cover most part of the chip in terms of resources for having more chances of having an upset. Both PB’s architectures where replicated 128 times. The number of replicas was chosen considering the resource utilization of the Vanilla design. With 128 replicas the Vanilla design covers close to a third of the chip, as
5. ARCHITECTURE UNDER TEST

A harness was designed to test the different architectures. The harness can be seen as a wrapper which connects to the PB variant. The harness is generic, i.e., PB variants can be replaced without modifying the harness. The main goal of the harness is to detect failures in the architectures. A side goal is to wrap the design and connect it to the interfaces.

The harness makes comparisons for each monitored signal between the 128 instantiated designs. The comparison is made bit by bit. The bit 1 of the bus A is compared with the bit 1 of the bus A of the other PBs. The process is performed...
across all the bus. When a mismatch is found, a '1' is written to a one shot register. Figure 5.4 shows the detector logic used in the harness. The logic was replicated for each bus.

Figure 5.3: Picoblaze vanilla layout.

Figure 5.4: Proposed failure detector architecture.

Two steps are involved in the detection of the error in a bus. First, AND and OR
operations are performed for the same bit across the 128 PBs. Four combinations can be made with the results of the AND and OR operations. The next step is the Error Detect phase, where the results of the OR and AND operations are compared. If both results are equal, then there is no fault in the bus. On the other hand, if the results are different, then an error is presented on the bus.

The harness is less than the 10% of the design. A small harness is desired so that it has a low probability of being hit with neutrons. To provide additional reliability to the harness, the detection stage of errors in the buses was triplicated; thus, having triplicated voters on each bus.

The outputs of the harness are the bits stored in the one-shot registers. Those bits indicate the presence of an upset in a certain bus. As the detectors are triplicated, the outputs of each bus are also three and they are written to three registers. The values stored in the one-shot registers are read with BSCAN and sent out of the architecture to a monitoring laptop.

The control computer is not under the beam. The tasks of the computer are to gather data from the test, to send commands to configure or reconfigure an FPGA, and to start and stop the tests. The data collected are stored in log files containing the initial time of the test, the architecture under test, the beam characteristics, the failures detected, reconfigurations performed, partial configurations performed and additional data with time stamps.

5.2 Medium Grain TMR

The second PB design is a Medium Grain (MG) TMR. The MGTMR design is a triplicated version of the Vanilla design with voters added throughout the architecture. The MGTMR was chosen to verify the proposed modeling of Fault Mitigation Techniques (FMTs) for not having to resynchronize as with a system
5.2. MEDIUM GRAIN TMR

redundancy or Coarse Grain TMR and for dividing an architecture into some portions, but not as many as with a Fine Grain TMR where 50 or more voters are added.

In the proposed modeling method it is important to identify the portions between voters, having an MGTMR gives enough flexibility to identify the area while significantly increasing the reliability. However, in the case of a large architecture not entirely custom made, it is challenging to get acquainted with it and identify the portions. Also, the insertion of more voters as in an FGTMR does not necessarily mean an increase in the reliability.

The block diagram of the MGTMR version with the PB soft processor is shown on Figure 5.5. It is noteworthy to point out that the Memory with scrubbing block was almost completely isolated by inserting voters on its signals. The goal was to simplify the endeavor of having shared resources between portions. In practice, the memory still has signals without voters on them, such as the reset and the clock. However, in the design, the clocks are triplicated from the beginning.

The Figure 5.5 shows that every module is triplicated along with the voters. The memory with scrubbing is isolated from the other logic of the soft processor. The microprocessor block represents the logic comprised by the ALU, scratchpad memory, registers, PC counter, stack, and control.

The resources of the microprocessor block can be divided into the ALU operations and the PC Counter operations. From the RTL of the architecture, it can be determined that roughly half of the resources are dedicated to each of the tasks. Moreover, the area of the portion with the BRAMs is half of the remaining resources. The architecture can be divided into three equal-sized portions for the model. However, a few resources are used in the soft processor for the PC counter and for the ALU calculations. Those resources are estimated to be scarce, around 5% of the total area.

The Vanilla Design and the MGTMR were radiated with neutrons to estimate their
reliability. The values of the Vanilla Design were used to gather evidence supporting the hypothesis of the thesis.

5.3 BLTMR Design

A third architecture was radiated with neutrons to estimate its cross section. The architecture is similar to the MGTMR; the difference is the addition of 43 majority voters in the architecture. The addition of the majority voters was automatically made by the BLTMR tool.

The BLTMR tool was made by BYU and Los Alamos National Lab to triplicate sensible structures of an FPGA architecture and if the resources allowed triplicate the whole architecture. The BLTMR tool searches for feedback loops, inserting voters on those loops at first. After using the tool in an architecture the output is a FGTMR version of the input architecture.

The total number of voters in the architecture are 139. The architecture is divided into smaller portions, partitioning the TMR domains to reduce the cross section. The goal of radiating this architecture is to find out if the addition of more majority voters
5.4 EXPERIMENTAL SETUP

to the MGTMR will lead to an increase or a decrease in the reliability.

5.4 Experimental Setup

The radiation test experiments for the three architectures were performed at the Los Alamos National Laboratory, in the Los Alamos Neutron Science Center (LANSCE)[66], New Mexico, USA. The total time under the beam for the tested architectures is roughly an hour.

The test was performed from November 16 to November 21, 2015. The test has a neutron energy spectrum from 0.1 MeV to 600 MeV [67]. Three KC705 evaluation boards were used during the test; swapping them from time to time.

Outside of the beam target zone, an external scrubber was placed. The external scrubber is connected through JTAG and has a scrubbing period of 2.54s. In addition, a computer was connected to give the commands and to read the data back from the BSCAN.

The BSCAN is slower than the complete execution of the program in the PBs. Allowing for the program to be executed multiple times between readbacks of the console. The one-shot registers enable the capability of storing the first fault presented in the architecture. Faults presented afterward are disregarded.

When a failure is detected through the BSCAN, the architecture is stopped and fully reconfigured; ensuring a proper operation from the beginning. When no failures are detected the output on the computer shows a time stamp with the number of readbacks. Moreover, the console shows that the readback is finished and the legend No errors found.

The output of the console when a failure exists contains the following information:
5. ARCHITECTURE UNDER TEST

- Time stamp of the readback.
- Frame, word and bit of the error.
- Number of errors detected.
- Total upset bits.
- Message when reconfiguration finishes.

A fault injection test is made before to get an idea of how much time each architecture should spend under the beam to present a significant amount of upsets. The results of fault injection give an idea of the number of upsets needed in the configuration memory to cause a failure. Likewise, with fault injection, the number of sensible bits can be counted.

Figure 5.6 presents the histogram for fault injecting the vanilla and MGTMR architectures. The histogram shows that more faults should be injected to the MGTMR design to get an upset. Table 5.1 shows the results.

<table>
<thead>
<tr>
<th>Design</th>
<th>Total injections</th>
<th>Average injections to failure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vanilla</td>
<td>131488</td>
<td>77.3459</td>
</tr>
<tr>
<td>MGTMR</td>
<td>5392145</td>
<td>3171.85</td>
</tr>
</tbody>
</table>

The fault injection test was performed with a tool developed at BYU. The tool is capable of monitoring signals, configure FPGAs and injecting faults in different ways. The injection for the test was random. In Figure 5.7 the results of injecting the Vanilla design are shown in a GUI. When compared with the layout, it is easy to realize that the sensitive bits are where the resources are used.

The results for the tests are presented in the next chapter.
5.4. EXPERIMENTAL SETUP

Figure 5.6: Picoblaze fault injection histogram (vanilla on the left, MGTMR on the right).

Figure 5.7: Picoblaze vanilla fault injection.
5. ARCHITECTURE UNDER TEST
6.1 Models Evaluation

The architectures tested were the Vanilla design and the MGTMR design. Both based on the PB soft processor. The MGTMR was designed by adding TMR and voters throughout the Vanilla design. The voters were added to increase the reliability of the TMR by dividing the total area into smaller portions.

The goal is to gather evidence on the possibility of estimating the reliability of a mitigated design – the MGTMR design for this case – by knowing the reliability of the unmitigated design and the resources between voters of the unmitigated design. With the experiments performed the neutron cross section of both architectures was determined. The cross section is dynamic as the architectures were operating at the time of the radiation. The cross section is the sensible area of the architecture.
While testing the designs, a computer is gathering data in log files. Those files are then processed to collect the information of interest. The log files are long files describing the upsets found, the errors, the address, the frame of the errors along with timestamps and messages useful for the tester for each readback.

After processing the data of 6 tests performed on the Vanilla design. The tests results for each run are shown in Table 6.1. The board column specifies which of the three boards was used during testing. Having multiple boards is a good practice to spot if a board has permanent damage. The adjacent fluence is used to compute the Fluence/Failure rate. The rate indicates the needed fluence of neutrons for a failure to happen in the architecture under test.

For the MGTMR design 6 tests were made. The tests results for each run are shown in Table 6.2. Three boards were also used on the MGTMR test. During some test no failure was found. The adjacent fluence of the last run was increased to make the architecture fail more often. With the increased fluence only 7 failures were presented. The increased fluence and the small number of failures in the architecture point toward a design with high reliability.

Table 6.3 shows the final results of the Vanilla and MGTMR design. A column representing the neutron dynamic cross section was added to the table. The cross section is the sensible area of the FPGA; it is computed as the reciprocal of the fluence
6.1. MODELS EVALUATION

Table 6.2: Results of MGTMR neutron testing.

<table>
<thead>
<tr>
<th>Test Date &amp; Time</th>
<th>Board</th>
<th>Fluence</th>
<th>Adj. Fluence</th>
<th>Failures</th>
<th>Fluence/Failure</th>
</tr>
</thead>
<tbody>
<tr>
<td>2015_11_17_20:46</td>
<td>1</td>
<td>2.07E+09</td>
<td>1.86E+09</td>
<td>1</td>
<td>1.86E+09</td>
</tr>
<tr>
<td>2015_11_18_06:30</td>
<td>2</td>
<td>8.79E+09</td>
<td>7.92E+09</td>
<td>2</td>
<td>3.96E+09</td>
</tr>
<tr>
<td>2015_11_19_09:09</td>
<td>0</td>
<td>5.00E+08</td>
<td>4.49E+08</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2015_11_19_10:09</td>
<td>0</td>
<td>7.07E+09</td>
<td>6.36E+09</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2015_11_19_14:59</td>
<td>2</td>
<td>1.39E+10</td>
<td>1.25E+10</td>
<td>2</td>
<td>6.25E+09</td>
</tr>
<tr>
<td>2015_11_20_20:04</td>
<td>0</td>
<td>3.09E+10</td>
<td>2.78E+10</td>
<td>7</td>
<td>3.97E+09</td>
</tr>
</tbody>
</table>

Table 6.3: Summarized results of Vanilla and MGTMR.

<table>
<thead>
<tr>
<th>Design</th>
<th>Improvement</th>
<th>Fluence</th>
<th>Adj. Fluence</th>
<th>Faults</th>
<th>Fluence/Failure</th>
<th>Cross Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vanilla</td>
<td>Baseline</td>
<td>2.27E+10</td>
<td>2.04E+10</td>
<td>87</td>
<td>2.35E+08</td>
<td>4.26E-09</td>
</tr>
<tr>
<td>MGTMR</td>
<td>20.18</td>
<td>6.33E+10</td>
<td>5.69E+10</td>
<td>12</td>
<td>4.74E+09</td>
<td>2.11E-10</td>
</tr>
</tbody>
</table>

to failure. The improvement in the cross section between the Vanilla design and the MGTMR means that the MGTMR is 20.18 times less sensible to failures; thus, the MGTMR is more reliable than the vanilla design.

The neutron dynamic cross section serves to compute a failure rate given a certain fluence. A fluence of $2.347 \times 10^5$ was assumed to compute the failure rates shown in both tables. The failure rate of the Vanilla was used on the proposed models. The experimental failure rate of the MGTMR is $4.9522 \times 10^{-5}$. Figure 6.1 shows the reliability curve for the MGTMR computed through equation 6.1.

$$R(t) = e^{-\lambda t}$$  \hspace{1cm} (6.1)

Figure 6.1 was used to verify the model by comparing it with both models.

For the first model, $\lambda_1 = \lambda_2 = \lambda_3$ as three equal parts were assumed. The sum of each failure rate gives the total failure rate, which is equal to 0.001 $\frac{\text{failure}}{\text{hour}}$. To further
detail the model, shared resources and different portions sizes were considered. The second model is a more detailed and general model. For the second model $\lambda_1 = \lambda_2 = \lambda_3$ which is equal to $0.001 \text{failure/hour}$. However, the modification lays in the addition of $\lambda_{c12}$, the parameter accounting for the failure rate of the shared resources between portion 1 and portion 2. As stated before, there are no other shared resources as the third portion, corresponding to the BRAM with scrubbing, is practically isolated from the other portions.

During the experiments, an external scrubber was connected. The scrubber completes a full readback and scrub cycle in 2.54 seconds. The time is converted into hours and plugged as the $\mu$ of the models.

The plots of the two models along with the experimental reliability are shown in Figure 6.2. The first model shows that the approximation to the experimental reliability curve is close. The plot of the first model runs all along above the experimental plot, meaning that the approach is optimistic but still close to the actual reliability. The curve decreases almost at the same rate as the experimental curve, showing that modeling the MGTMR architecture by dividing the cross section is a favorable approach.
6.2 MAJORITY VOTERS PLACEMENT

The second approach introduces the concept of sharing resources between partitions. When a particle hits a shared-resource module, the fault will have an effect on both partitions, making the architecture more prompt to failures. At the same time, the shared resources make the model more accurate. Even though the shared resources are small, their impact in the model is considerable.

The effect can be seen in the Figure 6.2. In red, the plot of the first model’s reliability is always on top of the experimental plot. In orange, the plot for the second model runs along the bottom part of the experimental curve – the model is pessimistic. The second plot is closer to the experimental curve, illustrating that introducing the shared resources in the model gives a favorable output.

![Figure 6.2: Proposed models and experimental reliability curve.](image)

6.2 Majority Voters Placement

The BLTMR architecture was also tested under the neutron beam. The architecture adds 129 voters to the MGTMR design. The voters were automatically added by the BLTMR tool, seeking feedback loops within the MGTMR design.
6. RESULTS

Table 6.4: Results of BLTMR neutron testing.

<table>
<thead>
<tr>
<th>Test Data &amp; Time</th>
<th>Board</th>
<th>Fluence</th>
<th>Adj. Fluence</th>
<th>Failures</th>
<th>Fluence/Failure</th>
</tr>
</thead>
<tbody>
<tr>
<td>2015_11_16 20:51</td>
<td>202</td>
<td>1.61E+09</td>
<td>1.45E+09</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2015_11_16 21:35</td>
<td>202</td>
<td>1.65E+09</td>
<td>1.48E+09</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2015_11_16 22:24</td>
<td>202</td>
<td>3.59E+09</td>
<td>3.24E+09</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2015_11_17 11:59</td>
<td>202</td>
<td>4.79E+08</td>
<td>4.32E+08</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2015_11_17 21:37</td>
<td>202</td>
<td>3.76E+10</td>
<td>3.39E+10</td>
<td>8</td>
<td>4.23E+09</td>
</tr>
<tr>
<td>2015_11_18 10:03</td>
<td>202</td>
<td>1.44E+10</td>
<td>1.30E+10</td>
<td>4</td>
<td>3.24E+09</td>
</tr>
<tr>
<td>2015_11_18 14:10</td>
<td>202</td>
<td>1.49E+10</td>
<td>1.34E+10</td>
<td>6</td>
<td>2.23E+09</td>
</tr>
<tr>
<td>2015_11_18 19:35</td>
<td>202</td>
<td>4.91E+10</td>
<td>4.43E+10</td>
<td>10</td>
<td>4.43E+09</td>
</tr>
<tr>
<td>2015_11_19 18:59</td>
<td>202</td>
<td>3.69E+10</td>
<td>3.33E+10</td>
<td>6</td>
<td>5.55E+09</td>
</tr>
</tbody>
</table>

The goal is to identify the effect of placing more majority voters in an architecture. The dynamic neutron cross section was determined with the experiment performed over the architecture. The results will be compared with the MGTMR to decide if the impact of more voters is positive or negative.

The setup of the experiment is the same as for the other two architectures. The results obtained after processing the data of 8 tests performed on the BLTMR design are shown in Figure 6.4. The adjacent fluence is the fluence which compensates for the distance to the beam and the thickness of the board between the device and the beam. The Fluence/Failure rate indicates the needed fluence of neutrons for a failure to happen in the architecture under test.

Table 6.5 shows the final results of the Vanilla, MGTMR design, and BLTMR. The column of cross section represents the sensible area of the FPGA; it is computed as the reciprocal of the fluence to failure. The improvement in the cross section between the Vanilla design and the BLTMR means that the BLTMR is 18.07 times less sensible to failures. The most interesting fact is that the BLTMR has a larger cross section than the MGTMR, even though the BLTMR is based on MGTMR but with additional
6.3. Remarks

Table 6.5: Summarized results of Vanilla, MGTMR and BLTMR.

<table>
<thead>
<tr>
<th>Design</th>
<th>Improvement</th>
<th>Fluence</th>
<th>Adj. Fluence</th>
<th>Faults</th>
<th>Fluence/Failure</th>
<th>Cross Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vanilla</td>
<td>Baseline</td>
<td>2.27E+10</td>
<td>2.04E+10</td>
<td>87</td>
<td>2.35E+08</td>
<td>4.26E-09</td>
</tr>
<tr>
<td>MGTMR</td>
<td>20.18</td>
<td>6.33E+10</td>
<td>5.69E+10</td>
<td>12</td>
<td>4.74E+09</td>
<td>2.11E-10</td>
</tr>
<tr>
<td>BLTMR</td>
<td>18.07</td>
<td>1.60E+11</td>
<td>1.44E+11</td>
<td>34</td>
<td>4.25E+09</td>
<td>2.35E-10</td>
</tr>
</tbody>
</table>

voters in feedback loops. The inclusion of additional voters may lead to more shared resources in the architecture. If seen from the modeling perspective, the addition of more voters represent an increase in states and transitions; incrementing the probability of transitioning to the failure state.

6.3 Remarks

The experiments performed under the beam yield a dynamic cross section of the architectures under test for each run. After processing the data the final cross section is known. The MGTMR architecture presents a decrease of 20.18 in cross section when compared against the unmitigated design. Making the MGTMR more reliable than the vanilla design.

For the theoretical estimation of the reliability, two models representing the behavior of the MGTMR were proposed. The first model divides the total failure rate in three equal parts. The second model adds a failure rate for shared resources between part 1 and part 2. The implication of considering shared resources is that the architecture becomes more prone to fail.

The reliability curve of the first model is close to the experimental estimation. The curve goes above the experimental and the uneven estimation; decreasing almost as the experimental curve does. The result supports that dividing the failure rate to estimate the reliability of a TMR is a favorable approach.
6. RESULTS

The second model introduces shared resources. The curve of the model is closer and below the experimental estimation. The addition of shared resources presents the effect of decreasing the reliability of the architecture at a higher rate when compared to the first model. The effect is a consequence of the fact that a fault on one of the shared-resources modules is equivalent to a fault on both modules.

For the experiments regarding voters placements, the results show that incrementing the number of voters does not always increase the reliability of the system. Placing voters is an important task that needs further research to establish a method capable of identifying the places to add voters in order to have the least amount of voters and the highest reliability on an architecture.
CONCLUSIONS AND FUTURE WORK

The use of SRAM-FPGAs is growing in critical applications. However, most architectures are not tested for its reliability. This work establishes a method to estimate the reliability of FPGA architectures with FMTs based on dividing the dynamic cross section of an unmitigated design. The division is made considering the portion of resources between voters of the mitigated design. The goal of the model is to estimate the reliability of the architecture accurately after applying fault mitigation techniques.

The use of Markov chains proofs useful for modeling the behavior of the architecture. Markov chains have the advantages of easily adding states and transitions to improve the similarity of the model to the architecture. Markov models are flexible for capturing the operation of the mitigated architecture.

The work supports that by characterizing an unmitigated design, it is possible to
compute the reliability after applying FMT. As expected, the introduction of voters in the architecture drastically improves the reliability of the system, as shown in the cross sections results.

The first model, with three evenly distributed portions, gave a similar output when compared to the experimental results. The approach is optimistic as the plot lies above the experimental results’ plot.

The second model improves the reliability estimation by adding shared resources in between the first and second portions. The addition of those modules gets the model closer to representing the reliability of the MGTMR architecture. By adding shared resources, the reliability curve presented a faster decrease as a fault on one of the shared-resources modules is equivalent to a fault on both modules.

The addition of the shared resources not only gives a closer approximation, the second model also succeeds in having a pessimistic approximation. A pessimistic approach that is close to the reality is favorable for critical applications. A risk factor is considered when designing avionics for spacecraft. When designing for meeting requirements dictated by the pessimistic approximation, the design will have better chances to cope with unfavorable situations.

On the other hand, it is interesting that the addition of more voters with the BLTMR tool does not improve the overall reliability. The results point in the direction of other researchers’ results that the reliability of an architecture is sensible to the placement of voters.

The work presented gathered evidence that estimations of the reliability can be made, reasonably accurate, by considering the logic between voters on a mitigated design. The model needs the cross section of an unmitigated design as input. Radiating only one design and having the flexibility to compute the reliability without any cost for several FMTs configurations can be translated into shorter time to deliver the final product and lower budget spent in testing the architectures. The work has a
positive impact on low-budget applications as only the unmitigated architecture should be radiated once; instead of radiating each proposed mitigated design.

7.1 Future Work

The proposed method gives a close estimation of the reliability in a mitigated architecture. Developing a tool to make the estimation of the reliability given the input parameters can extend the reach of the method. Industrial and academic projects, may benefit from the developed tool for the low-cost and short time to compute the reliability of their designs. Providing flexibility to test multiple configurations without an increase in the developing cost.

The method can also have a positive impact on other tools. Several tools exists to place voters and triplicate portions of an architecture automatically. Tools such as BLTMR and XTMR will benefit from adding the proposed method to give an estimation of the reliability once the design is mitigated. Having an estimation of the reliability as an output of the tools will allow shorter times to choose a suitable configuration for the mission.

For the voters placement, more tests should be performed to study the effect of placing of voters. Having more voters does not mean an increase of reliability for all cases. Testing more architectures with different voters-placing methods will provide a broaden the understanding of their effect in the reliability. The goal of the voter-placing methods is to maximize the reliability of the system while adding the least number of majority voters.
REFERENCES


REFERENCES


Workshop, Norfolk, USA, July 1999.


REFERENCES


REFERENCES


[58] Quinn, Heather M., Paul S. Graham, Michael J. Wirthlin, Brian Pratt, Keith S. Morgan, Michael P. Caffrey, and James B. Krone, “A test methodology for deter-


